BORLAND® C++

ADDENDUM

BORLAND

Borland C++

Borland C++ 3.0

Windows 3.1 API-Funktionen

BORLAND

Copyright © 1991 by Borland International, INC.
Jede Vervielfältigung dieses Handbuches sowie des Softwareprogrammes Borland C++ wird strafrechtlich verfolgt. Die Rechte an der Dokumentation und die Rechte an dem Softwareprogramm Borland C++ liegen bei Borland International.
Der rechtmäßige Erwerb der Programmdisketten und der Handbücher erlaubt die Nutzung der Programme analog der Benutzung eines Buches. Entsprechend der Unmöglichkeit, daß ein Buch zugleich an verschiedenen Orten von mehreren Personen gelesen wird, darf das Softwareprogramm Borland C++ nicht gleichzeitig von verschiedenen Personen an verschiedenen Orten und auf verschiedenen Geräten benutzt werden. Diskettenkopien dürfen lediglich zum Zweck der Datensicherung angefertigt werden.
Sie (der Lizenznehmer von Borland C++) dürfen eigene Programme, die Sie mit dem Compiler Borland C++ geschrieben und compiliert haben, ohne Einschränkung oder Verpiflichtung zu zusätzlichen Lizenzgebühren verwenden, weitergeben oder verkaufen, solange diese Programme einen Copyright-Vermerk enthalten. Unter Copyright-Vermerk verstehen wir entweder Ihren eigenen Copyright-Vermerk oder gegebenenfalls einen Hinweis wie »Created using Borland C++ by Borland International 1987, 88, 89, 90, 91«. Die Originaldisketten von Borland C++ beinhalten einige Dateien mit Hardware- und Zeichensatzinformationen, die von der Standard-Grafikbibliothek (GRAPHICS.LIB) verwendet werden. Diese Dateien mit den Suffixen CHR und BGI unterliegen besonderen Nutzungsbeschränkungen. Sie dürfen nicht als Dateien weiterverkauft werden. Sie dürfen diese Dateien aber in Ihren eigenen mit Borland C++ geschriebenen und compilierten Programmen verwenden. Sie dürfen diese compilierten Programme auch weiterverkaufen, solange sie einen Copyright-Vermerk tragen.
Die Beispielprogramme von Borland C++, die auf den Originaldisketten mitgeliefert werden, demonstrieren die verschiedenen Eigenschaften von Borland C++. Sie dienen ausschließlich zu Lehrzwecken. Sie dürfen diese Beispielprogramme editieren oder für den eigenen Gebrauch modifizieren. Sie dürfen diese Programme jedoch nicht als Ganzes oder in Teilen in compilierter Form oder als Quelltext weitergeben oder verkaufen. Sie dürfen die Routinen aus den Beispielprogrammen in eigenen Anwenderprogrammen verwenden, solange die resultierenden Programme nicht einem Beispielprogramm oder einem Teil eines Beispielprogrammes in Erscheinung oder Funktion gleichen.

Einschränkung der Gewährleistung
Es wird keine Garantie für die Richtigkeit des Inhaltes dieses Handbuches übernommen. Da sich Fehler, trotz aller Bemühungen, nie vollständig vermeiden lassen, sind wir für Hinweise jederzeit dankbar.

Herausgeber
Borland GmbH
Gautinger Straße 10
Postfach
8130 Starnberg
Telefon 0 18 51 / 2 63 - 0
Telefax 081 51 / 142 09

Übertragung ins Deutsche: Peter Baier, Roland Konstanty, Waltraud Kopelent, Arne Schäpers

Typographische Konzeption des Innenteils: Rudolf Paulus Gorbach

Grafiken: Andreas Höhne, Sid Gastl

Satz: Hanna Bergbauer

PostScript-Belichtung: CD Computer & Dokumentations GmbH, Neuler

Druck und Bindung: Schoder Druck GmbH, Gersthofen

Papier: Schleipen GmbH. Bad Dürkheim. Das verwendete Papier ist säurefrei und alterungsbeständig. Die Produktion erfolgt in einem geschlossenen Wasserkreislauf ohne Einsatz von optischen Aufhellern und unter Verwendung von chlorarm gebleichten, holzfreien Zellstoffqualitäten. Die Abwasserreinigung wird in einer betriebseigenen chemisch-biologischen Kläranlage vorgenommen.

2., unveränderte Auflage, Mai 1992

Inhalt

1. Bibliothek für Standarddialogfenster 1

Das Dialogfenster Farben 3
Die Farbmodelle des Dialogfensters Farben 4
Das Dialogfenster Farben zur Anzeige der Grundfarben verwenden 6
Das Dialogfenster Farben zur Anzeige selbstdefinierter Farben verwenden 7
Das Dialogfenster Schriftart 9
Das Dialogfenster Schriftart in Ihrer Anwendung anzeigen 10
Die Dialogfenster Datei öffnen und Datei speichern unter 12
Das Dialogfenster Datei öffnen in Ihrer Anwendung anzeigen 12
Das Dialogfenster Datei speichern unter in Ihrer Anwendung anzeigen 15
Listen in den Dialogfenstern Datei öffnen und Datei speichern unter überwachen 16
Dateinamen in den Dialogfenstern Datei öffnen und Datei speichern unter überwachen 17
Die Dialogfenster Drucken und Druckereinrichtung 18
Gerätetreiber und das Dialogfenster Drucken 18
Das Dialogfenster Drucken für den voreingestellten Drucker anzeigen 19
Die Dialogfenster Suchen und Ersetzen 20
Das Dialogfenster Suchen anzeigen 21
Das Dialogfenster Ersetzen anzeigen 22
Dialogfensterbotschaften für die Dialogfenster Suchen und Ersetzen bearbeiten 23
Standarddialogfenster anpassen 24
Geeignete und ungeeignete Anpassungen 24
Filterfunktionen und anwendungsdefinierte Dialogfensterschablonen 25
Das selbstdefinierte Dialogfenster anzeigen 28
Hilfe für das Standarddialogfenster bereitstellen 30
Fehlerüberwachung 31

2. Die DDE-Bibliothek (DDEML) 33

Grundlagen 34
Interaktion von Client und Server 34
Transaktionen und die DDE-Callback-Funktion 34
Servicenamen, Themennamen und Elementnamen 35
Das Thema System 35
Initialisierung 37
Callback- Funktion 38
String- Verwaltung 39
Servicenamen 41

Servicenamen registrieren 41
Servicenamenfilter 42
Konversationsverwaltung 42
Einzelkonversationen 43
Mehrfachkonversationen 46
Datenverwaltung 48
Transaktionsverwaltung 50
Die Request-Transaktion 51
Die Poke-Transaktion 51
Die Advise-Transaktion 52
Die Execute-Transaktion 53
Synchrone und asynchrone Transaktionen 53
Transaktionssteuerung 55
Transaktionsklassen 55
Übersicht 56
Fehlererkennung 58
Überwachende Anwendungen 58

3. OLE-Bibliotheken 63

Grundlagen der Objekt Grundlagen der Objektverknüpfung und -einbettung 63
Verbunddokumente 63
Verknüpfte und eingebettete Objekte 64
Vorteile der OLE 66
Zwischen OLE und DDEML wählen 67
Datentransfer unter OLE 70
Client-Anwendungen 70
Server-Anwendungen 70
Objekt-Handler 71
Kommunikation zwischen den OLE-Bibliotheken 71
Konventionen der Zwischenablage 72
Registrierung 75
Die Benutzerschnittstelle der Client-Anwendung 78
Die Benutzerschnittstelle der Server-Anwendung 81
Speicherformate für Objekte 82
Client- Anwendungen 84
Eine Client-Anwendung starten 85
Ein Verbunddokument öffnen 86
Dokumente verwalten 86
Ein Dokument speichern 87
Ein Dokument schließen 87
Asynchrone Operationen 88
Objekte anzeigen und drucken 90
Objekte öffnen und schließen 90
Objekte löschen 91
Die Client-Befehle Ausschneiden und Kopieren 91
Objekte erzeugen 93

Der Befehl Rückgängig 95
Der Befehl <Klassenname> Objekt 96
Der Befehl Verknüpfungen 96
Eine Client-Anwendung schließen 97
Server-Anwendungen 98
Eine Server-Anwendung starten 99
Ein Dokument oder ein Objekt öffnen 101
Die Server-Befehle Ausschneiden und Kopieren 102
Die Befehle Aktualisieren, Speichern unter und Neu 103
Eine Server-Anwendung schließen 104
Objekt-Handler 105
Objekt-Handler implementieren 106
Objekte mit einem Objekt-Handler erzeugen 108
DDE direkt anwenden 110
Client-Anwendungen und DDE 110
Server-Anwendungen und DDE 112
Konversationen 113
Elemente des Themas System 113
Standardnamen für Elemente und die Steuerung von Benachrichtigungen 114
Standardbefehle in ausführbaren DDE-Strings 116

4. Windows-Funktionen 121

5. Datentypen 485

6. Botschaften 493

Benachrichtigungsbotschaften 525

7. Strukturen 529

8. Makros 665

9. Escape-Codes 671

10. DDE-Transaktionen 675

11. Ereignisse und Botschaften des Datei-Managers 689

Datei-Manager-Ereignisse 689
Datei-Manager-Botschaften 691

12. Botschaften der Systemsteuerung 697

13. Botschaften der Standarddialogfenster 703

14. Botschaften installierbarer Treiber 709

15. Grafikdateiformate 717

Dateiformate von Bitmaps 717
Strukturen der Bitmap-Dateien 717
Bitmap-Komprimierung 719
Beispiel für ein Bitmap 721
Dateiformat von Symbolressourcen 722
Symbolverzeichnis 722
Symbolbilder 723
Symbolauswahl in Windows 724
Dateiformate von Mauszeigerressourcen 724
Das Mauszeigerverzeichnis 724
Das Mauszeigerbild 726
Mauszeigerauswahl in Windows 727

16. Das Dateiformat der Zwischenablage 729

Header der Zwischenablagedatei 729
Struktur der Zwischenablagedatei 729

17. Das Metafile-Format 731

Metafile-Header 731
Standard-Metafile-Record 732
Positionierbare Windows-Metafiles 734
Richtlinien für Windows-Metafiles 735
Beispiel einer Metafile-Programm-Ausgabe 735
Funktionsspezifische Metafile-Records 737

Index 751

1. Bibliothek für Standarddialogfenster

Standarddialogfenster erleichtern Ihnen das Entwickeln von Anwendungen für Microsoft Windows. Um ein Standarddialogfenster anzuzeigen, braucht eine Anwendung nur eine einzelne Funktion aufzurufen – sie muß keine Dialogfensterfunktion und keine Ressourendatei, die eine Dialogfensterschablone enthält, erzeugen. Die dynamische Linkbibliothek COMMDLG.DLL stellt eine Funktion und eine Schablone für jeden Standarddialogfenstertyp zur Verfügung. Alle Dialogfensterfunktionen bearbeiten Botschaften und Benachrichtigungen für ein Standarddialogfenster und dessen Steuerelemente. Eine Dialogfensterschablone definiert das Erscheinungsbild eines Standarddialogfensters und seiner Steuerelemente.

Zusätzlich zur Vereinfachung der Entwicklung von Windows-Anwendungen hilft ein Standarddialogfenster Benutzern, indem es die Durchführung bestimmter Operationen vereinheitlicht. Sobald Windows-Entwickler Standarddialogfenster in ihren Anwendungen benutzen, können Benutzer ähnliche Operationen in verschiedenen Anwendungen mit Hilfe identischer Steuerelemente durchführen.

Dieses Kapitel beschreibt die verschiedenen Standarddialogfenster und enthält Quelltextbeispiele, die Ihnen bei der Verwendung von Standarddialogfenstern in Ihren Windows-Anwendungen behilflich sein sollen.

Die folgende Liste zeigt die Standarddialogfenstertypen in der Reihenfolge, in der sie in diesem Kapitel beschrieben werden:

Name	Beschreibung
Farben	Zeigt die verfügbaren Farben an, aus denen der Benutzer auswählen kann. Stellt Steuerelemente bereit, mit denen der Benutzer selbst eine Farbe definieren kann.
Schriftart	Zeigt Schriften, Punktgrößen und Farben an, die den verfügbaren Schriftarten entsprechen; nachdem der Benutzer eine Schriftart auswählt, zeigt das Dialogfenster Beispieltext in dieser Schrift an.
Datei öffnen	Zeigt eine Liste der Dateinamen, die den angegebenen Dateinamenserweiterungen, Verzeichnissen und Laufwerken entsprechen. Durch das Auswählen eines dieser Dateinamen zeigt der Benutzer an, welche Datei von der Anwendung geöffnet werden soll.

Name	Beschreibung
Datei speichern unter	Zeigt eine Liste der Dateinamen, die den angegebenen Dateinamenserweiterungen, Verzeichnissen und Laufwerken entsprechen. Durch das Auswählen eines dieser Dateinamen zeigt der Benutzer an, welche Datei von der Anwendung gespeichert werden soll.
Drucken	Zeigt Informationen über den installierten Drucker und dessen Konfiguration. Mit Hilfe der Steuerelemente in diesem Dialogfenster kann der Benutzer festlegen, wie Ausgaben gedruckt werden sollen. Außerdem kann er den Druckvorgang starten.
Druckereinrichtung	Zeigt die Liste der aktuell verfügbaren Drucker. Der Benutzer kann einen Drucker aus dieser Liste auswählen. Das Dialogfenster stellt außerdem Optionen zum Einstellen der Papierausrichtung, -größe und -zufuhr bereit, wenn der ausgewählte Drucker diese Optionen unterstützt. Zusätzlich zum direkten Aufruf kann das Dialogfenster *Druckereinrichtung* auch im Dialogfenster *Drucken* geöffnet werden.
Suchen	Zeigt ein Editierfeld an, in dem der Benutzer einen String eingeben kann, nach dem die Anwendung suchen soll. Der Benutzer kann die Richtung der Suche angeben, ob die Suche die Groß- und Kleinschreibung berücksichtigen soll und ob der String ein ganzes Wort ist.
Ersetzen	Zeigt zwei Editierfelder an, in die der Benutzer Strings eingeben kann. Der erste String bezeichnet das Wort oder den Wert, den die Anwendung ersetzen soll. Der zweite String bezeichnet das Wort oder den Wert, durch den ersetzt werden soll.

Anwendungen, die Standarddialogfenster verwenden, sollten mindestens 8 KByte für die Stackgröße angeben, wie im folgenden Beispiel gezeigt wird:

```
NAME cd

EXETYPE WINDOWS

STUB    'WINSTUB.EXE

CODE    PRELOAD MOVEABLE DISCARDABLE

DATA    PRELOAD MOVEABLE MULTIPLE

HEAPSIZE  1024

STACKSIZE 8192

EXPORTS
  FILEOPENHOOKJPROC @  1
```

Das Dialogfenster Farben

Das Dialogfenster *Farben* enthält Steuerelemente, mit denen ein Benutzer Farben auswählen und erzeugen kann.

Die folgende Abbildung zeigt ein solches Dialogfenster:

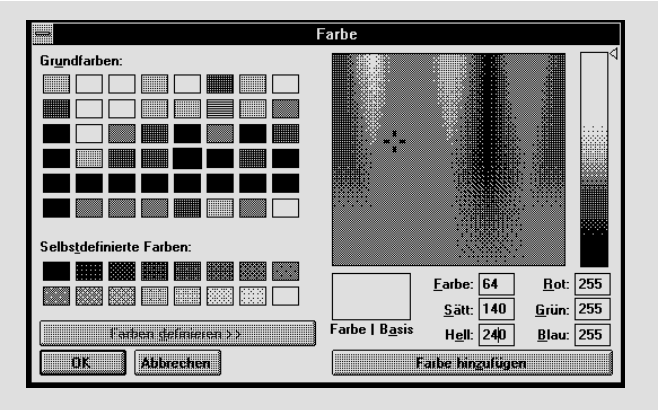


Abbildung 1.1 Das Dialogfenster Farben

Das Steuerelement *Grundfarben* zeigt bis zu 48 Farben an. Die tatsächliche Anzahl der angezeigten Farben wird vom Bildschirmtreiber bestimmt. Beispielsweise zeigt ein VGA-Treiber 48 Farben, ein monochromer Treiber nur 16 Farben an. Mit dem Steuerelement *Grundfarben* kann der Benutzer eine angezeigte Farbe auswählen.

Um das Steuerelement *Selbstdefinierte Farben* zu füllen, muß der Benutzer den Schalter *Farben definieren* anklicken. Das Steuerelement zeigt dann die selbstdefinierten Farben an. Der Benutzer kann eines der 16 Rechtecke in diesem Steuerelement auswählen und danach eine neue Farbe mit Hilfe folgender Methoden erzeugen:

- Den Rot-, Grün- und Blauwert (RGB) in den Editierfeldern *Rot*, *Grün* und *Blau* angeben und dann den Schalter *Hinzufügen* auswählen, um die neue Farbe im gewählten Rechteck anzuzeigen.

- Den Mauszeiger zum Steuerelement *Farbpalette* oben im Dialogfenster bewegen, um die Werte für den Farbton und die Sättigung auszuwählen; den Mauszeiger in das Steuerelement *Helligkeit* rechts vom Steuerelement *Farbpalette* bewegen und dann den Schalter *Hinzufügen* wählen, um die neue Farbe im Rechteck anzuzeigen.

- Die Werte für Farbton, Sättigung und Helligkeit (HSL) in den Editierfeldern *Farbe*, *Sätt* und *Hell* angeben und dann den Schalter *Hinzufügen* wählen, um die neue Farbe im Rechteck anzuzeigen.

Das Steuerelement *Farbe|Basis* zeigt die Misch- und die Grundfarben, die der Auswahl des Benutzers entsprechen. Eine Mischfarbe wird durch die Kombination verschiedener Grundfarben erzeugt. Das Element *Flags* der *CHOOSECOLOR*-

Struktur enthält ein Flag, das einen *Hilfe*-Schalter anzeigt, wenn es gesetzt ist. Weitere Informationen finden Sie in der Beschreibung der *CHOOSECOLOR*-Struktur in Kapitel 7 »Strukturen«.

Eine Anwendung kann das Dialogfenster *Farben* vollständig oder teilweise geöffnet anzeigen. Wenn es teilweise geöffnet angezeigt wird, kann der Benutzer die selbstdefinierten Farben nicht ändern.

Die Farbmodelle des Dialogfensters Farben

Das Dialogfenster *Farben* verwendet zwei Modelle zum Festlegen von Farben: das RGB- und das HSL-Modell.

Das Farbmodell RGB

Das RGB-Modell wird verwendet, um Farben für Bildschirmanzeigen und andere Geräte, die Licht ausstrahlen, festzulegen. Gültige Werte für Rot, Grün und Blau liegen im Bereich von 0 bis 255, wobei 0 die geringste und 255 die höchste Intensität darstellt.

Die folgende Abbildung zeigt, wie die Grundfarben Rot, Grün und Blau kombiniert werden können, um vier zusätzliche Farben zu erzeugen. Bei Bildschirmgeräten ergibt sich die Farbe Schwarz, wenn die Werte für Rot, Grün und Blau auf 0 gesetzt werden. Bildschirmtechnisch gesehen ist Schwarz also die Abwesenheit aller Farben.

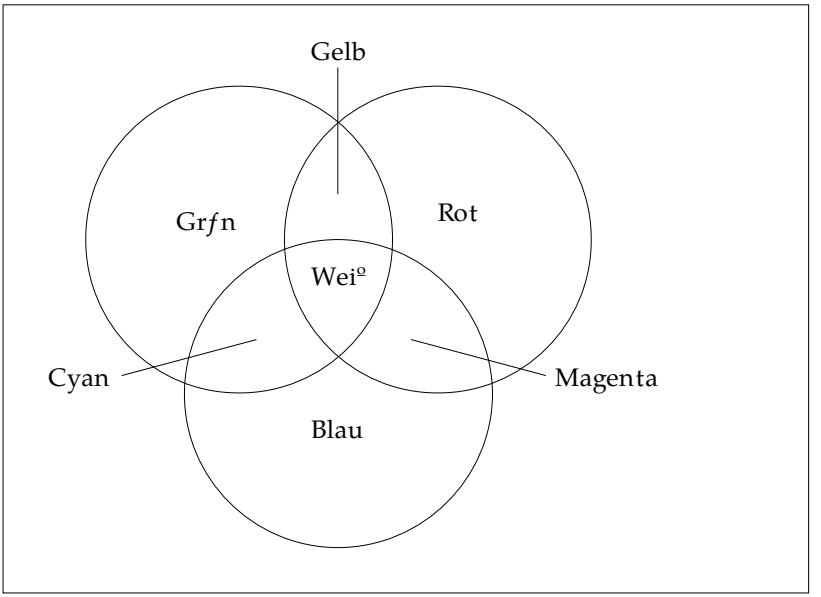


Abbildung 1.2 Kombination der Grundfarben im RGB-Modell

Die folgende Aufstellung zeigt acht Farben und die ihnen zugeordneten RGB-Werte:

Farbe	RGB-Werte
Rot	255, 0, 0
Grün	0, 255, 0
Blau	0, 0, 255
Cyan	0, 255, 255
Magenta	255, 0, 255
Gelb	255, 255, 0
Weiß	255, 255, 255
Schwarz	0, 0, 0

Windows speichert Farben intern als 32-Bit-RGB-Werte. Das höherwertige Byte des höherwertigen *WORD* ist reserviert, das niederwertige Byte des höherwertigen *WORD* gibt die Intensität der Blau-Komponente an. Das höherwertige Byte des niederwertigen *WORD* gibt die Intensität der Grün-Komponente an und das niederwertige Byte des niederwertigen *WORD* die Intensität der Rot-Komponente.

Das Farbmodell HSL

Das Dialogfenster *Farben* enthält Steuerelemente zur Festlegung der HSL-Werte. Die folgende Abbildung zeigt die Steuerelemente *Farbpalette* und *Helligkeit* in diesem Dialogfenster sowie die Wertebereiche, die der Benutzer mit diesen Steuerelementen festlegen kann.

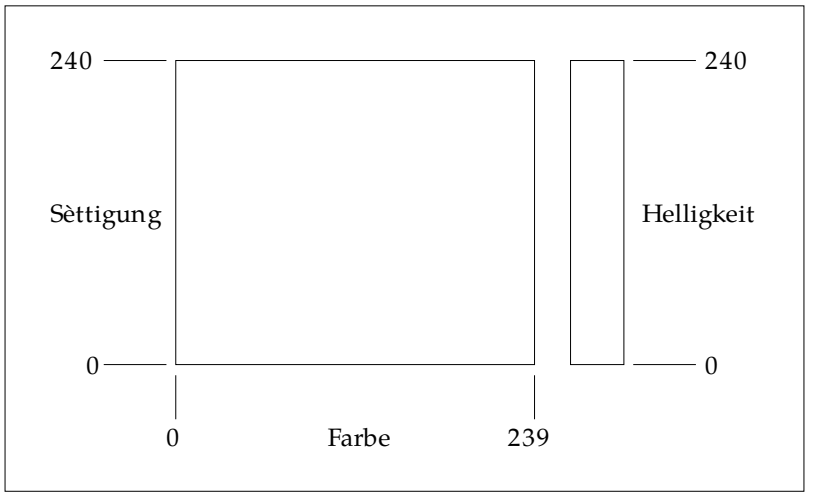


Abbildung 1.3 Farbauswahl im HSL-Modell

Im Dialogfenster *Farben* müssen die Werte für Sättigung und Helligkeit im Bereich von 0 bis 240 und der Farbwert im Bereich von 0 bis 239 liegen.

HSL- in RGB-Werte konvertieren

Die in der Bibliothek COMMDLG.DLL bereitgestellte Dialogfensterfunktion für das Dialogfenster *Farben* enthält Code, der HSL- in die entsprechenden RGB-Werte konvertiert. Die folgende Aufstellung zeigt mehrere Farben mit den entsprechenden HSL- und RGB-Werten:

Farbe	HSL-Werte	RGB-Werte
Rot	(0, 240, 120)	(255, 0, 0)
Gelb	(40, 240, 120)	(255, 255, 0)
Grün	(80, 240, 120)	(0, 255, 0)
Cyan	(120, 240, 120)	(0, 255, 255)
Blau	(160, 240, 120)	(0, 0, 255)
Magenta	(200, 240, 120)	(255, 0, 255)
Weiß	(0, 0, 240)	(255, 255, 255)
Schwarz	(0, 0, 0)	(0, 0, 0)

Das Dialogfenster Farben zur Anzeige der Grundfarben verwenden

Eine Anwendung kann das Dialogfenster *Farben* anzeigen, damit ein Benutzer eine Farbe aus der Liste der Bildschirmfarben wählen kann. Dieser Abschnitt beschreibt, welcher Quelltext und welche Strukturen in Ihrer Anwendung dies ermöglichen.

Die CHOOSECOLOR-Struktur initialisieren

Bevor Sie das Dialogfenster *Farben* anzeigen, müssen Sie eine *CHOOSECOLOR*-Struktur initialisieren. Diese Struktur sollte global sein oder als Variable vom Typ **static** deklariert werden. Die Elemente dieser Struktur enthalten folgende Informationen:

- die Größe der Struktur;
- das Fenster, dem das Dialogfenster zugeordnet ist;
- ob die Anwendung das Standarddialogfenster anpaßt;
- die Filterfunktion und die anwendungsdefinierte Dialogfensterschablone, die für eine angepaßte Version des Dialogfensters *Farben* verwendet werden;
- die RGB-Werte der gewählten Grundfarbe.

Wenn Ihre Anwendung das Dialogfenster nicht anpaßt und Sie wollen, daß der Benutzer eine einzelne Farbe aus den Grundfarben wählen kann, sollten Sie die *CHOOSECOLOR*-Struktur in der folgenden Art und Weise initialisieren:

```
/* Farbvariablen */

CHOOSECOLOR cc;
COLORREF clr;
COLORREF aclrCust[16];
int i;
```

```
/* Steuerelemente für selbstdefinierte Farben
   mit Weiß initialisieren. */

for (i = 0; i < 16; i++)
    aclrCust[i] = RGB(255, 255, 255);

/* clr mit Schwarz initialisieren. */

clr = RGB(0, 0, 0);

/* Alle Felder der Struktur mit Null initialisieren. */
memset(&cc, 0, sizeof(CHOOSECOLOR));

/* Die benötigten Elemente der CHOOSECOLOR-Struktur initialisieren. */

cc.lStructSize = sizeof(CHOOSECOLOR);
cc.hwndOwner = hwnd;
cc.rgbResult = clr;
cc.lpCustColors = aclrCust;
cc.Flags = CC_PREVENTFULLOPEN;

if (ChooseColor(&cc))
  . /* Mit Hilfe von cc.rgbResult die vom Benutzer
  .    angeforderte Farbe wählen */
```

In diesem Beispiel enthält das Array, auf das das Element *lpCustColors* zeigt, 16 *DWORD*-RGB-Werte, die die Farbe Weiß festlegen. Das Flag *CC_PREVENT-FULLOPEN* ist im Element *Flags* gesetzt, um die Auswahl einer selbstdefinierten Farbe durch den Benutzer zu verhindern.

Die Funktion ChooseColor aufrufen

Nach der Initialisierung der Struktur sollten Sie die Funktion *ChooseColor* aufrufen. Wenn die Funktion erfolgreich ist und der Benutzer den Schalter *OK* wählt, um das Dialogfenster zu schließen, enthält das Element *rgbResult* den RGB-Wert der vom Benutzer gewählten Grundfarbe.

Das Dialogfenster Farben zur Anzeige selbstdefinierter Farben verwenden

Eine Anwendung kann das Dialogfenster *Farben* anzeigen, damit der Benutzer eine selbstdefinierte Farbe erzeugen und auswählen kann. Dieser Abschnitt beschreibt, welcher Quelltext und welche Strukturen in Ihrer Anwendung dies ermöglichen.

Die CHOOSECOLOR-Struktur initialisieren

Bevor Sie das Dialogfenster Farbe anzeigen, müssen Sie eine *CHOOSECOLOR*-Struktur initialisieren, die global sein oder als Variable vom Typ **static** deklariert werden sollte. Die Elemente dieser Struktur enthalten folgende Informationen:

- die Größe der Struktur;
- das Fenster, dem das Dialogfenster zugeordnet ist;

- ob die Anwendung das Standarddialogfenster anpaßt;
- die Filterfunktion und die anwendungsdefinierte Dialogfensterschablone, die für eine angepaßte Version des Dialogfensters *Farben* verwendet werden;
- die RGB-Werte der gewählten Grundfarbe.

Wenn Ihre Anwendung das Dialogfenster nicht anpaßt und Sie wollen, daß der Benutzer selbstdefinierte Farben wählen kann, sollten Sie die *CHOOSECOLOR*-Struktur in der folgenden Art und Weise initialisieren:

```
/* Farbvariablen */

CHOOSECOLOR chsclr;
DWORD dwCustClrs[16]= { RGB(255, 255, 255), RGB(239, 239, 239),
                        RGB(223, 223, 223), RGB(207, 207, 207),
                        RGB(191, 191, 191), RGB(175, 175, 175),
                        RGB(159, 159, 159), RGB(143, 143, 143),
                        RGB(127, 127, 127), RGB(111, 111, 111),
                        RGB(95, 95, 95),    RGB(79, 79, 79),
                        RGB(63, 63, 63),    RGB(47, 47, 47),
                        RGB(31, 31, 31),    RGB(15, 15, 15)
                      };
BOOL fSetColor = FALSE;
int i;

chsclr.lStructSize = sizeof(CHOOSECOLOR);
chsclr.hwndOwner = hwnd;
chsclr.hInstance = NULL;
chsclr.rgbResult = 0L;
chsclr.lpCustColors = (LPWORD) dwCustClrs;
chsclr.Flags = CC_FULLOPEN;
chsclr.lCustData = 0L;
chsclr.lpfnHook = (FARPROC) NULL;
chsclr.lpTemplateName = (LPSTR) NULL;
```

In diesem Beispiel enthält das Array, auf das das Element *lpCustColors* zeigt, 16 RGB-Werte, die 16 Grautöne angeben und jeweils 32 Bit lang sind. Das Flag *CC_FULLOPEN* ist im Element *Flags* gesetzt, damit das Dialogfenster vollständig angezeigt wird.

Die Funktion ChooseColor aufrufen

Nach der Initialisierung der Struktur sollten Sie die Funktion *ChooseColor* aufrufen, wie im folgenden Quelltextfragment gezeigt:

```
if (fSetColor = ChooseColor(&chsclr))
.
./ * Mit Hilfe von chsclr.lpCustColors die vom Benutzer
.    angegebene Farbe auswählen. */
```

Wenn die Funktion erfolgreich ist und der Benutzer den Schalter *OK* wählt, um das Dialogfenster zu schließen, zeigt das Element *lpCustColors* auf ein Array mit den RGB-Werten der vom Benutzer angeforderten selbstdefinierten Farben.

Anwendungen können die Benutzung selbstdefinierter Farben stärker beeinflussen, indem sie einen neuen Botschaftsbezeichner für den von der Konstanten *COLOROKSTRING* definierten String erzeugen. Die Anwendung erzeugt den neuen Botschaftsbezeichner, indem sie die Funktion *RegisterWindowMessage* aufruft und die Konstante als einzigen Parameter übergibt. Nach diesem Aufruf erhält die Anwendung unmittelbar vor dem Schließen des Dialogfensters eine Botschaft. Der Parameter *lParam* dieser Botschaft enthält einen Zeiger auf die *CHOOSECOLOR*-Struktur. Die Anwendung kann das Element *lpCustColors* dieser Struktur verwenden, um die aktuelle Farbe zu überprüfen. Wenn die Anwendung einen Wert ungleich Null zurückgibt, während sie diese Botschaft bearbeitet, wird das Dialogfenster nicht geschlossen.

Entsprechend können Anwendungen einen neuen Botschaftsbezeichner für den durch die Konstante *SETRGBSTRING* definierten String erzeugen. Die Filterfunktion der Anwendung kann den Botschaftsbezeichner verwenden, der durch den Aufruf der Funktion *RegisterWindowMessage* mit der Konstante *SETRGBSTRING* zurückgegeben wird, um eine Farbe im Dialogfenster zu initialisieren.

Die folgende Quelltextzeile setzt z.B. die Farbauswahl auf Blau:

```
SendMessage(hwndDlg, wSetRGBMsg, 0, (LPARAM) RGB(0, 0, 255));
```

In diesem Beispiel ist *wSetRGBMsg* der von der Funktion *RegisterWindowMessage* zurückgelieferte Botschaftsbezeichner. Der Parameter *lParam* der Funktion *SendMessage* wird auf die RGB-Werte der gewünschten Farbe gesetzt. Der Parameter *wParam* wird nicht verwendet.

Die Anwendung kann in diesem Aufruf von *SendMessage* jeden gültigen RGB-Wert angeben. Wenn die RGB-Werte mit einer der Grundfarben übereinstimmen, wählt das System die Grundfarbe und aktualisiert die Steuerelemente *Farbpalette* und *Helligkeit*. Wenn die RGB-Werte mit keiner der Grundfarben übereinstimmen, aktualisiert das System zwar die Steuerelemente *Farbpalette* und *Helligkeit*, die Grundfarbenauswahl bleibt jedoch unverändert.

Beachten Sie, daß das System das Dialogfenster nicht aktualisiert, wenn dieses nicht vollständig geöffnet ist und die Anwendung RGB-Werte sendet, die mit keiner der Grundfarben übereinstimmen. Dies ist auch nicht nötig, da die Steuerelemente *Farbpalette* und *Helligkeit* nicht sichtbar sind, wenn das Dialogfenster nur teilweise geöffnet ist.

Weitere Informationen über das Bearbeiten registrierter Fensterbotschaften finden Sie im Abschnitt »Die Dialogfenster *Suchen* und *Ersetzen*«.

Das Dialogfenster Schriftart

Das Dialogfenster *Schriftart* enthält Steuerelemente, mit deren Hilfe ein Benutzer eine Schrift, eine Schriftauszeichnung (z.B. fett, kursiv oder normal), eine Punktgröße und einen Effekt (z.B. Unterstrichen, Durchgestrichen oder eine Textfarbe) auswählen kann. Die folgende Abbildung 1.4 zeigt das Dialogfenster *Schriftart*.

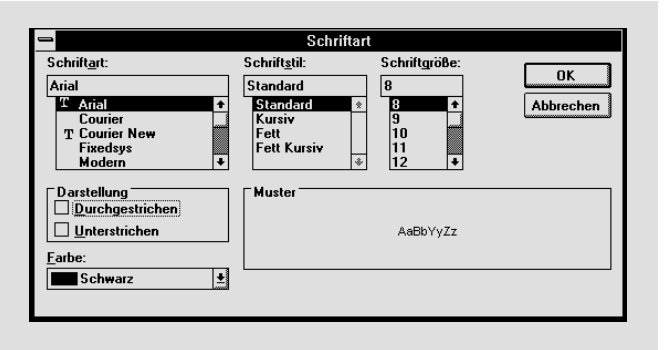


Abbildung 1.4
Das Dialogfenster Schriftart

Das Dialogfenster Schriftart in Ihrer Anwendung anzeigen

Das Dialogfenster *Schriftart* erscheint, nachdem Sie die Elemente einer *CHOOSEFONT*-Struktur initialisiert und die Funktion *ChooseFont* aufgerufen haben. Die Struktur sollte global sein oder als Variable vom Typ **static** deklariert werden. Die Elemente von *CHOOSECOLOR*-Strukturen enthalten folgende Informationen:

- die Attribute der Schrift, die anfänglich im Dialogfenster angezeigt werden soll;
- die Attribute der Schrift, die der Benutzer gewählt hat;
- die Punktgröße der Schrift, die der Benutzer gewählt hat;
- ob die Schriftenliste einem Drucker, einem Bildschirm oder beiden entspricht;
- ob nur TrueType-Schriften verfügbar sind;
- ob das Feld *Effekte* im Dialogfenster erscheinen soll;
- ob Dialogfensterbotschaften von einer anwendungsunterstützten Filterfunktion bearbeitet werden sollen;
- ob die Punktgrößen der auswählbaren Schriften auf einen festgelegten Bereich beschränkt werden sollen;
- ob das Dialogfenster nur WYSIWIG-Schriften (What You See Is What You Get) anzeigen soll (diese Schriften sind sowohl im Drucker- als auch im Bildschirmspeicher resident);
- die Farbe, mit der die Funktion *ChooseColor* Text im Beispiel-Fenster anzeigt, wenn die Anwendung das Dialogfenster erstmals darstellt.
- die Farbe, die der Benutzer für Textausgaben gewählt hat.

Um das Dialogfenster *Schriftart* anzuzeigen, sollte eine Anwendung folgende Schritte durchführen:

- wenn die Anwendung Druckerschriften benötigt, ein Gerätekontext-Handle für den Drucker abrufen und dem Element *hDC* der *CHOOSEFONT*-Struktur dieses Handle zuweisen (zeigt das Dialogfenster *Schriftart* nur Bildschirmschriften an, sollte dieses Element auf *NULL* gesetzt werden);
- die geeigneten Flags im Element *Flags* der *CHOOSEFONT*-Struktur setzen (entweder *CF_SCREENFONTS*, *CF_PRINTERFONTS* oder *CF_BOTH*);
- das Element *rgbColors* der *CHOOSEFONT*-Struktur setzen, wenn die voreingestellte Farbe (Schwarz) nicht geeignet ist;
- das Element *nFontType* der *CHOOSEFONT*-Struktur mit einer geeigneten Konstanten initialisieren;
- die Elemente *nSizeMin* und *nSizeMax* der *CHOOSEFONT*-Struktur setzen, wenn der Wert *CF_LIMITSIZE* im Element *Flags* angegeben ist;
- die Funktion *ChooseFont* aufrufen.

Das folgende Beispiel initialisiert die *CHOOSEFONT*-Struktur und ruft die Funktion *ChooseFont* auf:

```
LOGFONT lf;
CHOOSEFONT cf;

/* Alle Elemente der Struktur mit Null initialisieren. */

memset(&cf, 0, sizeof(CHOOSEFONT));

cf.lStructSize = sizeof(CHOOSEFONT);
cf.hwndOwner = hwnd;
cf.lpLogFont = &lf;
cf.Flags = CF_SCREENFONTS | CF_EFFECTS;
cf.rgbColors = RGB(0, 255, 255); /* hellblau */
cf.nFontType = SCREEN_FONTTYPE;

ChooseFont(&cf);
```

Wenn der Benutzer das Dialogfenster *Schriftart* schließt, indem er den *OK*-Schalter wählt, liefert die Funktion *ChooseColor* Informationen über die ausgewählte Schrift in der *LOGFONT*-Struktur zurück, auf die das Element *lpLogFont* zeigt. Eine Anwendung kann diese *LOGFONT*-Struktur verwenden, um die Schrift für Textausgaben auszuwählen.

Das folgende Beispiel wählt eine Schrift unter Verwendung der *LOGFONT*-Struktur aus und gibt einen Text-String aus:

```
hdc = GetDC(hwnd);
hFont = CreateFontIndirect(cf.lpLogFont);
hFontOld = SelectObject(hdc, hFont);
TextOut(hdc, 50, 150,
   "AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz", 52);
SelectObject(hdc, hFontOld);
DeleteObject(hFont);
ReleaseDC(hwnd, hdc);
```

Eine Anwendung kann auch die *WM_CHOOSEFONT_GETLOGFONT*-Botschaft verwenden, um die aktuelle *LOGFONT*-Struktur für das Dialogfenster *Schriftart* abzurufen, bevor dieses vom Benutzer geschlossen wird.

Die Dialogfenster Datei öffnen und Datei speichern unter

Die Dialogfenster *Datei öffnen* und *Datei speichern unter* sind in ihrer Erscheinung ähnlich. Sie enthalten Steuerelemente, mit denen ein Benutzer die Position und den Namen einer oder mehrerer Dateien angeben kann. Im Fall des Dialogfensters *Datei öffnen* wählt der Benutzer die Datei, die geöffnet werden soll. Im Dialogfenster *Datei speichern unter* wählt der Benutzer die Datei, die gespeichert werden soll.

Das Dialogfenster Datei öffnen in Ihrer Anwendung anzeigen

Das Dialogfenster *Datei öffnen* erscheint, nachdem Sie die Elemente einer *OPENFILENAME*-Struktur initialisiert und die Funktion *GetOpenFileName* aufgerufen haben. Die folgende Abbildung zeigt das Dialogfenster *Datei öffnen*.

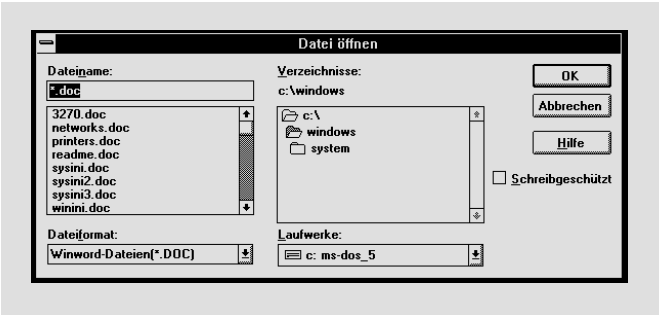


Abbildung 1.5 Das Dialogfenster Datei öffnen

Vor dem Aufruf von *GetOpenFileName* enthalten die Strukturelemente Verzeichnisnamen und Filter, die im Dialogfenster verwendet werden sollen. Ein Filter ist eine Dateinamenserweiterung. Die Dialogfensterfunktionen verwenden diese Namenserweiterung, um entsprechende Dateinamen aus einem Verzeichnis herauszufiltern. Nach dem Aufruf enthalten die Strukturelemente unter anderem den Namen der gewählten Datei und die Anzahl der Zeichen im Dateinamen.

Um ein *Öffnen*-Dialogfenster anzuzeigen, sollte eine Anwendung folgende Schritte durchführen:

- die gültigen Filter in einem Zeichen-Array speichern;
- dem Element *lpstrFilter* einen Zeiger auf dieses Array zuweisen;

- dem Element *nFilterIndex* den Wert des Index des voreingestellten Filters zuweisen;
- dem Element *lpstrFile* einen Zeiger auf ein Array zuweisen, das den anfänglichen Dateinamen enthält und den gewählten Dateinamen aufnimmt;
- dem Element *nMaxFile* den Wert zuweisen, der die Länge des Array für den Dateinamen angibt.
- das Element *lpstrFileTitle* mit dem Zeiger auf einen Puffer initialisieren, der den Namen der gewählten Datei im Titelfeld der Dateiauswahlliste aufnimmt;
- das Element *nMaxFileTitle* setzen, um die Länge des Puffers anzugeben;
- dem Element *lpstrInitialDir* einen Zeiger auf einen String zuweisen, der das anfängliche Verzeichnis angibt (wenn dieses Element nicht auf einen gültigen String zeigt, muß es auf 0 gesetzt werden oder auf einen *NULL*-String zeigen);
- dem Element *lpstrTitle* einen Zeiger auf einen String zuweisen, der den Namen angibt, der in der Titelzeile des Dialogfensters erscheinen soll (wenn dieser Zeiger *NULL* ist, ist der Titel *Öffnen*);
- dem Element *lpstrDefExt* einen Zeiger auf einen String mit der voreingestellten Namenserweiterung zuweisen (diese Namenserweiterung kann 0, 1, 2 oder 3 Zeichen lang sein);
- die Funktion *GetOpenFileName* aufrufen.

Das folgende Beispiel initialisiert eine *OPENFILENAME*-Struktur, ruft die Funktion *GetOpenFileName* auf und öffnet die Datei mit Hilfe des Strukturelements *lpstrFile*. Die *OPENFILENAME*-Struktur sollte global sein oder als Variable vom Typ **static** deklariert werden.

```
OPENFILENAME ofn;
char szDirName[256];
char szFile[256], szFileTitle[256];
UINT i, cbString;
char chReplace;    /* Trennzeichen für den String in szFilter */
char szFilter[256];
HFILE hf;

/* Namen des Systemverzeichnisses ermitteln und in
   szDirName speichern. */

GetSystemDirectory(szDirName, sizeof(szDirName));
szFile[0] = '\0';

if ((cbString = LoadString(hinst, IDS_FILTERSTRING,
       szFilter, sizeof(szFilter))) == 0) {
    ErrorHandler();
    return 0L;
}
chReplace = szFilter[cbString - 1]; /* Platzhalterzeichen abrufen */
for (i = 0; szFilter[i] != '\0'; i++) {
    if (szFilter[i] == chReplace)
```

```
        szFilter[i] = '\0';
}

/* Alle Elemente der Struktur mit Null initialisieren. */

memset(&ofn, 0, sizeof(OPENFILENAME));

ofn.lStructSize = sizeof(OPENFILENAME);
ofn.hwndOwner = hwnd;
ofn.lpstrFilter = szFilter;
ofn.nFilterIndex = 1;
ofn.lpstrFile = szFile;
ofn.nMaxFile = sizeof(szFile);
ofn.lpstrFileTitle = szFileTitle;
ofn.nMaxFileTitle = sizeof(szFileTitle);
ofn.lpstrInitialDir = szDirName;
ofn.Flags = OFN_SHOWHELP | OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST;

if (GetOpenFileName(&ofn)) {
    hf = _lopen(ofn.lpstrFile, OF_READ);
        .
        . /* Dateioperationen durchführen. */
        .
}
else
    ErrorHandler();
```

Der String, auf den die Konstante *IDS_FILTERSTRING* im vorhergehenden Beispiel verweist, wird folgendermaßen in der Ressourcendefinitionsdatei definiert:

```
STRINGTABLE
BEGIN
 IDS_FILTERSTRING "Write-Dateien(*.WRI)|*.wri|" +
                  "Word-Dateien(*.DOC)|*.doc|"
END
```

Die vertikalen Striche in diesem String sind Platzhalterzeichen. Nachdem der String durch die Funktion *LoadString* abgerufen worden ist, werden die Platzhalterzeichen durch *NULL* ersetzt. Platzhalter kann jedes eindeutige Zeichen sein; es muß als letztes Zeichen im String angegeben werden. Das Initialisieren von Strings auf diese Art und Weise garantiert, daß die Teile des String aufeinanderfolgend im Speicher stehen und daß der String mit zwei Nullzeichen abgeschlossen wird.

Anwendungen, die Dateien über ein Netz öffnen können, können einen neuen Botschaftsbezeichner für den von der Konstanten *SHAREVISTRING* definierten String erzeugen. Dazu müssen sie die Funktion *RegisterWindowMessage* aufrufen und diese Konstante als einzigen Parameter übergeben. Danach wird die Anwendung immer dann benachrichtigt, wenn eine Zugriffsverletzung während eines Aufrufs der Funktion *OpenFile* auftritt. Weitere Informationen über das Verarbeiten registrierter Fensterbotschaften finden Sie im Abschnitt »Die Dialogfenster *Suchen* und *Ersetzen*«.

Das Dialogfenster Datei speichern unter in Ihrer Anwendung anzeigen

Das Dialogfenster *Datei speichern unter* erscheint, nachdem Sie die Elemente einer *OPENFILENAME*-Struktur initialisiert und die Funktion *GetSaveFileName* aufgerufen haben. Die folgende Abbildung zeigt das Dialogfenster *Datei speichern unter*.

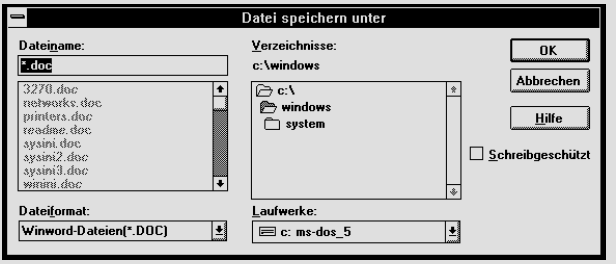


Abbildung 1.6 Das Dialogfenster Datei speichern unter

Vor dem Aufruf von *GetSaveFileName* enthalten die Strukturelemente den Namen des anfänglichen Verzeichnisses und einen Filter-String. Nach dem Aufruf enthalten sie den Namen der zu speichernden Datei und die Anzahl der Zeichen dieses Dateinamens.

Das folgende Beispiel initialisiert eine *OPENFILENAME*-Struktur, ruft die Funktion *GetSaveFileName* auf und speichert die Datei. Die *OPENFILENAME*-Struktur sollte global sein oder als **static**-Variable deklariert werden.

```
OPENFILENAME ofn;
char szDirName[256];
char szFile[256], szFileTitle[256];
UINT i, cbString;
char chReplace;                    /* Trennzeichen für den String
                                      in szFilter. */
char szFilter[256];
HFILE hf;

/*
 * Namen des Systemverzeichnisses abrufen und in
 * szDirName speichern.
 */

GetSystemDirectory(szDirName, sizeof(szDirName));

if ((cbString = LoadString(hinst, IDS_FILTERSTRING,
       szFilter, sizeof(szFilter))) == 0) {
    ErrorHandler();
    return 0;
}

chReplace = szFilter[cbString - 1];
```

```
/* Platzhalterzeichen abrufen */

for (i = 0; szFilter[i] != '\0'; i++) {
    if (szFilter[i] == chReplace)
        szFilter[i] = '\0';
}

/* Alle Elemente der Struktur mit Null initialisieren. */

memset(&ofn, 0, sizeof(OPENFILENAME));

/* Die Elemente von OPENFILENAME initialisieren. */

szFile[0] = '\0';

ofn.lStructSize = sizeof(OPENFILENAME);
ofn.hwndOwner = hwnd;
ofn.lpstrFilter = szFilter;
ofn.lpstrFile = szFile;
ofn.nMaxFile = sizeof(szFile);
ofn.lpstrFileTitle = szFileTitle;
ofn.nMaxFileTitle = sizeof(szFileTitle);
ofn.lpstrInitialDir = szDirName;
ofn.Flags = OFN_SHOWHELP | OFN_OVERWRITEPROMPT;

if (GetSaveFileName(&ofn)) {
    .
    . /* Dateioperationen durchführen. */
    .
}
else
    ErrorHandler();
```

Der String, auf den die Konstante *IDS_FILTERSTRING* in diesem Beispiel verweist, wird in der Ressourcendefinitionsdatei definiert. Sie wird genauso verwendet, wie die Konstante *IDS_FILTERSTRING*, die im Abschnitt »Das Dialogfenster *Datei öffnen* in Ihrer Anwendung anzeigen« beschrieben wird.

Listen in den Dialogfenstern Datei öffnen und Datei speichern unter überwachen

Eine Anwendung kann die Benutzerauswahlen in den Dateiauswahllisten überwachen, um Daten in anwendungsdefinierten Steuerelementen zu bearbeiten und anzuzeigen. Eine Anwendung kann z.B. ein anwendungsdefiniertes Steuerelement verwenden, um die Gesamtgröße (in Byte) aller im Feld *Dateiname* ausgewählten Dateien anzuzeigen.

Eine Methode zur Ermittlung dieses Werts ist, die Gesamtanzahl der Byte immer wieder erneut zu berechnen, sobald der Benutzer eine Datei aus- oder abwählt. Eine schnellere Methode ist die Verwendung der *LBSELCHSTRING*-Botschaft, um eine neue Auswahl zu identifizieren und die entsprechende Dateilänge dem Wert hinzuzufügen, der im anwendungsdefinierten Steuerelement

angezeigt wird. Beachten Sie, daß das anwendungsdefinierte Steuerelement in diesem Beispiel ein Windows-Standardsteuerelement ist, das Sie in einer Ressourcendateischablone für eines der Standarddialogfenster bezeichnen.

Eine Anwendung registriert die Auswahländerungsbotschaft mit der Funktion *RegisterWindowMessage*. Wenn die Anwendung die Botschaft registriert hat, verwendet sie den Rückgabewert der Funktion, um Botschaften des Dialogfensters zu ermitteln. Die Botschaft wird in der anwendungsunterstützten Filterfunktion für das Standarddialogfenster bearbeitet. Der Parameter *wParam* jeder Botschaft bezeichnet die Liste, in der ausgewählt worden ist. Das niederwertige *WORD* des Parameters *lParam* bezeichnet den Listeneintrag. Das höherwertige *WORD* von *lParam* kann einen der folgenden Werte annehmen:

Wert	Bedeutung
CD_LBSELCHANGE	Gibt an, daß der vom niederwertigen *WORD* von *lParam* bezeichnete Eintrag ein Eintrag in einer Liste für Einfachauswahl ist.
CD_LBSELSUB	Gibt an, daß der vom niederwertigen *WORD* von *lParam* bezeichnete Eintrag nicht mehr in einer Liste für Mehrfachauswahl ausgewählt ist.
CD_LBSELADD	Gibt an, daß der vom niederwertigen *WORD* von *lParam* bezeichnete Eintrag in einer Liste für Mehrfachauswahl ausgewählt wurde.
CD_LBSELNOITEMS	Gibt an, daß keine Einträge in einer Liste für Mehrfachauswahl vorhanden sind.

Im Abschnitt »Die Dialogfenster *Suchen* und *Ersetzen*« finden Sie ein Beispiel für die Registrierung einer Standarddialogfensterbotschaft.

Dateinamen in den Dialogfenstern Datei öffnen und Datei speichern unter überwachen

Anwendungen können die normale Arbeit der Dialogfenster *Datei öffnen* und *Datei speichern unter* verändern, indem sie überwachen, welche Dateinamen der Benutzer eingibt und andere eindeutige Operationen durchführen. Eine Anwendung könnte z.B. das Schließen des Dialogfensters durch den Benutzer verhindern, wenn der gewählte Dateiname ungültig ist; eine andere Anwendung könnte es dem Benutzer ermöglichen, mehrere Dateinamen auszuwählen.

Um Dateinamen zu überwachen, sollte eine Anwendung die Botschaft *FILEOKSTRING* registrieren, indem sie die Funktion *RegisterWindowMessage* aufruft und als einzigen Parameter den Botschaftsnamen übergibt. Nachdem die Botschaft registriert ist, wird sie von der Dialogfensterfunktion in COMMDLG.DLL verwendet, um anzuzeigen, daß der Benutzer einen Dateinamen ausgewählt und den *OK*-Schalter gewählt hat und daß das Dialogfenster den Dateinamen überprüft hat und bereit ist, zur aufrufenden Funktion zurückzukehren. Die

Dialogfensterfunktion zeigt diese Aktionen durch das Senden der Botschaft an die Filterfunktion der Anwendung an. Nach Erhalt der Botschaft sollte die Filterfunktion einen Wert an die aufrufende Dialogfensterfunktion zurückgeben. Wenn die Filterfunktion die Botschaft nicht bearbeitet hat, sollte sie 0 zurückgeben; wenn sie die Botschaft bearbeitet hat und das Dialogfenster geschlossen werden soll, sollte die Funktion 0 zurückgeben. Hat die Filterfunktion die Botschaft bearbeitet und soll das Dialogfenster nicht geschlossen werden, sollte sie 1 zurückgeben. Alle anderen Rückgabewerte sind reserviert.

Die Dialogfenster Drucken und Druckereinrichtung

Mit den Steuerelementen des Dialogfensters *Drucken* kann ein Benutzer einen Drucker für einen bestimmten Druckauftrag konfigurieren. Er kann unter anderem die Druckqualität, den Seitenbereich und die Anzahl der Kopien festlegen. Die möglichen Einstellungen sind vom verwendeten Drucker abhängig. Die folgende Abbildung zeigt das Dialogfenster *Drucken*.

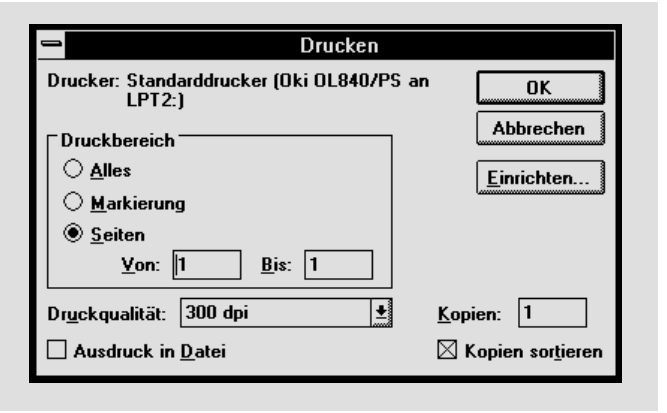


Abbildung 1.7 Das Dialogfenster Drucken

Der Schalter *Einrichten* im Dialogfenster *Drucken* öffnet das Dialogfenster *Druckereinrichtung*, in diesem Beispiel für einen PostScript-Drucker.

Das Dialogfenster *Druckereinrichtung* enthält Steuerelemente, mit denen der Benutzer die Druckereinstellungen ändern kann. Abbildung 1.8 zeigt dieses Fenster.

Gerätetreiber und das Dialogfenster Drucken

Das Dialogfenster *Drucken* unterscheidet sich von anderen Standarddialogfenstern dadurch, daß sich ein Teil seiner Dialogfensterfunktion in COMMDLG.DLL befindet und ein Teil in einem Druckertreiber. Ein Druckertreiber ist ein Programm, das einen Drucker konfiguriert, Befehle der grafischen

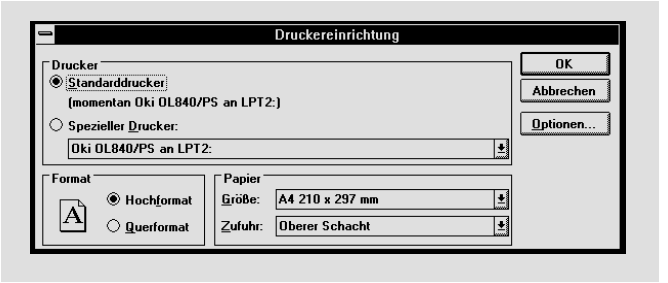


Abbildung 1.8 Das Dialogfenster Druckereinrichtung

Geräteschnittstelle (GDI) in Druckerbefehle konvertiert und Befehle für einen bestimmten Druckauftrag in der Druckwarteschlange speichert.

Ein Druckertreiber exportiert eine Funktion namens *ExtDeviceMode*, die ein Dialogfenster und seine Steuerelemente anzeigt. In früheren Versionen von Windows mußten Anwendungen die Funktion *LoadLibrary* aufrufen, um einen Gerätetreiber zu laden und die Funktion *GetProcAddress*, um die Adresse der Funktion *ExtDeviceMode* zu erhalten. Dies ist bei der Standarddialogfenster-Schnittstelle von Windows nicht mehr notwendig. Anstatt *LoadLibrary* und *GetProcAddress* aufzurufen, kann eine Anwendung eine einzige Funktion, *PrintDlg*, aufrufen, um das Dialogfenster *Drucken* anzuzeigen und den Druckauftrag zu starten. Die Funktion *PrintDlg* befindet sich in COMMDLG.DLL. Das beim Aufruf der Funktion *PrintDlg* angezeigte Dialogfenster unterscheidet sich von dem Dialogfenster, das direkt durch den Gerätetreiber aufgerufen wird. Die Funktionsweise ist trotz der unterschiedlichen Erscheinung sehr ähnlich.

Das Dialogfenster Drucken für den voreingestellten Drucker anzeigen

Um ein Dialogfenster *Drucken* für den voreingestellten Drucker anzuzeigen, muß eine Anwendung eine *PRINTDLG*-Struktur initialisieren und dann die Funktion *PrintDlg* aufrufen.

Die Elemente der *PRINTDLG*-Struktur können folgende Informationen enthalten:

- den Gerätekontext des Druckers;
- die Werte, die in den Steuerelementen des Dialogfensters erscheinen sollen;
- die Filterfunktion und die anwendungsdefinierte Dialogfensterschablone, die für eine angepaßte Version der Dialogfenster *Drucken* oder *Druckereinrichtung* verwendet werden;

Eine Anwendung kann ein Dialogfenster *Drucken* für den aktuell installierten Drucker anzeigen, indem sie die folgenden Schritte durchführt:

- das Flag *PD_RETURNDC* im Element *Flags* der *PRINTDLG*-Struktur setzen (dieses Flag sollte nur gesetzt werden, wenn die Anwendung ein Gerätekontext-Handle benötigt);
- die Elemente *lStructSize*, *hDevMode* und *hDevNames* initialisieren;

- die Funktion *PrintDlg* aufrufen und einen Zeiger auf die gerade initialisierte *PRINTDLG*-Struktur übergeben.

Das Setzen des *PD_RETURNDC*-Flag veranlaßt *PrintDlg*, das Dialogfenster *Drucken* anzuzeigen und ein Handle im Element *hDC* der *PRINTDLG*-Struktur zurückzugeben, das einen Gerätekontext für den Drucker bezeichnet. Die Anwendung übergibt das Handle als ersten Parameter an GDI-Funktionen, die Ausgaben auf dem Drucker vornehmen.

Das folgende Beispiel initialisiert eine *PRINTDLG*-Struktur und ruft die Funktion *PrintDlg* auf, bevor die Ausgabe gedruckt wird. Diese Struktur sollte global sein oder als **static**-Variable deklariert werden.

```
PRINTDLG pd;

/* Alle Elemente der Struktur mit Null initialisieren. */

memset(&pd, 0, sizeof(PRINTDLG));

/* Die benötigten Elemente der PRINTDLG-Struktur initialisieren. */

pd.lStructSize = sizeof(PRINTDLG);
pd.hwndOwner = hwnd;
pd.Flags = PD_RETURNDC;

/* Bei Erfolg eine Testseite drucken. */

if (PrintDlg(&pd) != 0) {
    Escape(pd.hDC, STARTDOC, 8, "Test-Doc", NULL);

    /* Text und Rechteck ausdrucken. */

    TextOut(pd.hDC, 50, 50, "Testseite für PrintDlg", 23);
    Rectangle(pd.hDC, 50, 90, 625, 105);
    Escape(pd.hDC, NEWFRAME, 0, NULL, NULL);
    Escape(pd.hDC, ENDDOC, 0, NULL, NULL);
    DeleteDC(pd.hDC);
    if (pd.hDevMode != NULL)
        GlobalFree(pd.hDevMode);
    if (pd.hDevNames != NULL)
        GlobalFree(pd.hDevNames);
}
else
    ErrorHandler();
```

Die Dialogfenster Suchen und Ersetzen

Die Dialogfenster *Suchen* und *Ersetzen* sind in ihrer Erscheinung ähnlich. Sie können mit dem Dialogfenster *Suchen* die Fähigkeit zur String-Suche in Ihre Anwendung aufnehmen. Das Dialogfenster *Ersetzen* ermöglicht auch das Ersetzen von Strings.

Das Dialogfenster Suchen anzeigen

Das Dialogfenster *Suchen* enthält Steuerelemente, mit deren Hilfe ein Benutzer folgende Angaben machen kann:

- den String, den die Anwendung suchen soll;
- ob der String ein ganzes Wort oder den Teil eines Worts angibt;
- ob die Anwendung Groß- und Kleinschreibung berücksichtigen soll;
- die Richtung, in der die Anwendung suchen soll (vor oder nach der aktuellen Position des Cursor);
- ob die Anwendung die Suche wiederholen soll, um nach dem nächsten Vorkommen des String zu suchen.

Die folgende Abbildung zeigt ein *Suchen*-Dialogfenster.

Abbildung 1.9 Das Dialogfenster Suchen

Um das Dialogfenster *Suchen* anzuzeigen, müssen Sie eine *FINDREPLACE*-Struktur initialisieren und die Funktion *FindText* aufrufen. Die Elemente dieser Struktur enthalten folgendes:

- zu welchem Fenster das Dialogfenster gehört;
- wie die Anwendung die Suche durchführen soll;
- einen Zeichenpuffer, der den String aufnehmen soll.

Um die *FINDREPLACE*-Struktur zu initialisieren, müssen folgende Aufgaben durchgeführt werden:

- das Element *lStructSize* mit Hilfe des Operators **sizeof** setzen;
- das Element *hwndOwner* mit dem Handle initialisieren, das das besitzende Fenster des Dialogfensters bezeichnet;
- wenn Sie das Dialogfenster *Suchen* ändern wollen, müssen Sie das Element *hInstance* setzen, damit es die Instanz des Moduls bezeichnet, das Ihre selbstdefinierte Dialogfensterschablone enthält;
- das Element *Flags* setzen, um den Auswahlstatus der Dialogfensteroptionen anzuzeigen (z.B. deaktiviert das Setzen des Flag *FR_NOUPDOWN* die Schalter *Aufwärts* und *Abwärts*, das Setzen des Flag *FR_NOWHOLEWORD* das Auswahlfeld *Als Wort* und das Setzen des Flag *FR_NOMATCHCASE* das Auswahlfeld *Groß-/Kleinschreibung*).

Wenn Sie eine selbstdefinierte Dialogfensterschablone oder Filterfunktion bereitstellen, müssen Sie zusätzliche Flags im Element *Flags* setzen:

- dem Element *lpstrFindWhat* einen Zeiger auf einen Puffer zuweisen, der den zu suchenden String aufnimmt;
- das Element *wFindWhatLen* setzen, um die Größe des Puffers in Byte anzugeben, auf den *lpstrFindWhat* zeigt;
- das Element *lCustData* mit selbstdefinierten Daten initialisieren, auf die die Anwendung möglicherweise zugreifen muß;
- wenn Ihre Anwendung das Dialogfenster *Suchen* anpaßt, müssen Sie dem Element *lpfnHook* einen Zeiger auf Ihre Filterfunktion zuweisen;
- wenn Ihre Anwendung eine selbstdefinierte Dialogfensterschablone verwendet, initialisieren Sie das Element *lpTemplateName* mit einem Zeiger auf den String, der die Schablone bezeichnet.

Das folgende Beispiel initialisiert die *FINDREPLACE*-Struktur und ruft dann die Funktion *FindText* auf. Die Struktur sollte global sein oder als Variable vom Typ **static** deklariert werden.

```
FINDREPLACE fr;

/* Alle Elemente der Struktur mit Null initialisieren */

memset(&fr, 0, sizeof(FINDREPLACE));

fr.lStructSize = sizeof(FINDREPLACE);
fr.hwndOwner = hwnd;
fr.lpstrFindWhat = szFindWhat;
fr.FindWhatLen = sizeof(szFindWhat);

hDlg = FindText(&fr);

break;
```

Das Dialogfenster Ersetzen anzeigen

Das Dialogfenster *Ersetzen* ist dem Dialogfenster *Suchen* ähnlich, hat jedoch kein Auswahlfeld zur Richtungsangabe und drei zusätzliche Steuerelemente, mit deren Hilfe der Benutzer folgende Angaben machen kann:

- den Ersatz-String;
- ob die Anwendung den String ersetzen soll, der aktuell hervorgehoben ist;
- ob die Anwendung alle Vorkommen des String ersetzen soll.

Abbildung 1.10 zeigt das Dialogfenster *Ersetzen*. Um das Dialogfenster *Ersetzen* anzuzeigen, müssen Sie eine *FINDREPLACE*-Struktur initialisieren und die Funktion *ReplaceText* aufrufen.

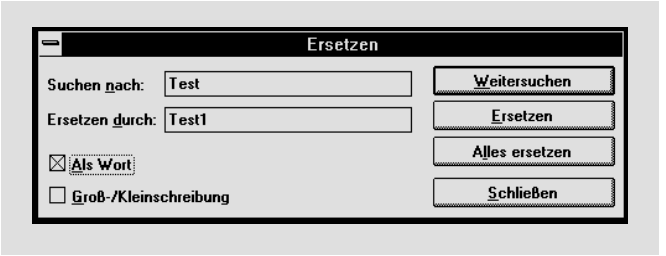


Abbildung 1.10 Das Dialogfenster Ersetzen

Dialogfensterbotschaften für die Dialogfenster Suchen und Ersetzen bearbeiten

Die Dialogfenster *Suchen* und *Ersetzen* unterscheiden sich von den anderen Standarddialogfenstern in zweierlei Hinsicht: erstens sind sie nicht-modal und zweitens senden ihre jeweiligen Dialogfensterfunktionen Botschaften an die Anwendung, die die Funktionen *FindText* oder *ReplaceText* aufruft. Diese Botschaften enthalten Daten, die vom Benutzer in den Steuerelementen der Dialogfenster angegeben werden, wie z.B. die Richtung, in der die Anwendung nach einem String suchen soll, ob zwischen Groß- und Kleinschreibung unterschieden werden soll und ob die Anwendung nach Übereinstimmung mit dem gesamten String suchen soll.

Um Botschaften der Dialogfenster *Suchen* und *Ersetzen* zu bearbeiten, muß eine Anwendung die eindeutige Botschaft des Dialogfensters registrieren, *FINDMSGSTRING*.

Die Anwendung registriert diese Botschaft mit der Funktion *RegisterWindowMessage* und verwendet den Rückgabewert der Funktion, um Botschaften von diesen beiden Dialogfenstern zu ermitteln.

Das folgende Beispiel registriert die Botschaft mit der Funktion *RegisterWindowMessage*:

```
UINT uFindReplaceMsg;

/* Die FindReplace-Botschaft registrieren. */

uFindReplaceMsg = RegisterWindowMessage(FINDMSGSTRING);
```

Nachdem die Anwendung diese Botschaft registriert hat, kann sie Botschaften für die Dialogfenster *Suchen* und *Ersetzen* bearbeiten, indem sie den Rückgabewert von *RegisterWindowMessage* verwendet. Das folgende Beispiel bearbeitet Botschaften für das Dialogfenster *Suchen* und ruft dann eine eigene *SearchFile*-Funktion auf, um den Text-String zu lokalisieren. Wenn der Benutzer das Dialogfenster schließt – also das Element *Flags* der *FINDREPLACE*-Struktur gleich der Konstanten *FR_DIALOGTERM* ist –, sollte das Handle ungültig gemacht werden und die Funktion Null zurückliefern.

```
LRESULT CALLBACK MainWndProc(HWND hwnd, UINT msg, WPARAM wParam,
    LPARAM lParam)
{
    FINDREPLACE FAR* lpfr;
```

```
    if (msg == uFindReplaceMsg) {
        lpfr = (FINDREPLACE FAR*) lParam;
        SearchFile((BOOL) (lpfr->Flags & FR_DOWN),
            (BOOL) (lpfr->Flags & FR_MATCHCASE));
        return 0;
}
```

Standarddialogfenster anpassen

Standarddialogfenster können den Anforderungen bestimmter Windows-Anwendungen angepaßt werden. Die Anpassung kann sehr umfangreich sein und das Verbergen der ursprünglichen Steuerelemente, das Hinzufügen neuer Steuerelemente oder eine Änderung der Größe des Originaldialogfensters einschließen. Sie kann aber auch einfach in der Anpassung eines einzelnen Steuerelements bestehen.

Entwickler müssen zur Anpassung eines Standarddialogfensters eine spezielle Filterfunktion und in den meisten Fällen auch eine selbstdefinierte Dialogfensterschablone bereitstellen. Anpassungen dieser Art erfordern eine beträchtliche Menge zusätzlichen Quelltexts – das Anzeigen eines selbstdefinierten Standarddialogfensters ist wesentlich aufwendiger als das Initialisieren der Elemente einer Struktur und das Aufrufen einer einzigen Funktion.

Anwendungen, die Steuerelemente in einem Standarddialogfenster unterklassifizieren, müssen dies während der Bearbeitung der *WM_INITDIALOG*-Botschaft in ihrer Filterfunktion tun. Dadurch erhält die Anwendung steuerelementspezifische Botschaften zuerst, weil sie das Steuerelement unterklassifiziert, nachdem das Standarddialogfenster seine Unterklassifizierungsfunktionen installiert hat. Die vorhergehende Filterfunktion sollte für alle Botschaften aufgerufen werden, die nicht von der Unterklassenfunktion der Anwendung bearbeitet werden, wie dies sonst bei Unterklassifizierungen geschieht.

Eine Anwendung kann kein Steuerelement unterklassifizieren, indem sie eine lokale Klasse definiert, um einen bestimmten Steuerelementtyp zu überschreiben. In diesem Fall würde das Datensegment nicht richtig initialisiert werden, wenn die Klasse aufgerufen würde – das Datensegment würde das Datensegment des Standarddialogfensters und nicht das der Anwendung sein.

Geeignete und ungeeignete Anpassungen

Aus der Sicht des Benutzers ist der größte Vorteil des Standarddialogfensters seine von Anwendung zu Anwendung gleichbleibende Erscheinung und Funktionsweise. Hierfür ist es wichtig, daß ein Entwickler ein Standarddialogfenster nur anpaßt, wenn es für eine Anwendung unbedingt erforderlich ist. Andernfalls gehen die konsistente Erscheinung und die einfache Programmierschnittstelle verloren. Geeignete Anpassungen lassen möglichst viele der ursprüng-

lichen Steuerelemente unberührt. Das Vergrößern des Dialogfensters oder das Hinzufügen neuer Steuerelemente an einer verfügbaren Position im Dialogfenster ist eine ungeeignete Anpassung. Das Verbergen originaler Steuerelemente oder anderweitiges Ändern der Funktionsweise originaler Steuerelemente ist ebenfalls eine ungeeignete Anpassung.

Filterfunktionen und anwendungsdefinierte Dialogfensterschablonen

Jedes Standarddialogfenster verwendet die in COMMDLG.DLL bereitgestellten Dialogfensterfunktionen und Dialogfensterschablonen. Eine Dialogfensterfunktion bearbeitet Botschaften und Benachrichtigungen für das Standarddialogfenster und dessen Steuerelemente. Die Dialogfensterschablone definiert die Erscheinung des Dialogfensters: seine Abmessungen, seine Position und die Abmessungen und Positionen seiner Steuerelemente.

Zusätzlich zu der bereitgestellten Dialogfensterfunktion und Dialogfensterschablone benötigt ein anwendungsdefiniertes Dialogfenster eine von Ihnen bereitgestellte Filterfunktion und normalerweise auch eine selbstdefinierte Version der Dialogfensterschablone.

Die Filterfunktion

Die in COMMDLG.DLL für Standarddialogfenster bereitgestellte Dialogfensterfunktion ruft die Filterfunktion der Anwendung auf, wenn diese das entsprechende Flag und einen Zeiger in der Struktur für das Standarddialogfenster setzt. Die Struktur eines Standarddialogfensters enthält ein Element *Flags*, das angibt, ob die Anwendung eine Filterfunktion bereitstellt. Die Struktur enthält außerdem das Element *lpfnHook*, das auf die Filterfunktion zeigt, wenn eine solche existiert. Wenn die Anwendung das Element *Flags* setzt, um die Existenz einer Filterfunktion anzuzeigen, muß sie auch das Element *lpfnHook* setzen. Das folgende Beispiel setzt die Elemente *Flags* und *lpfnHook* einer *OPENFILENAME*-Struktur, um die Filterfunktion einer Anwendung zu unterstützen:

```
#define STRICT

#include <windows.h>        /* wird für alle Windows-Anwendungen
                               benötigt */
#include <commdlg.h>
#include <string.h>
#include "header.h"         /* spezielle Header-Datei für dieses
                               Programm */

OPENFILENAME ofn;

   /* Den Namen des Systemverzeichnisses ermitteln und in
      szDirName speichern. */

    GetSystemDirectory((LPSTR)szDirName, 255);

    szFile[0] = '\0';
    ofn.lStructSize = sizeof(OPENFILENAME);
```

```
ofn.hwndOwner = hwnd;
ofn.hInstance = hInst;
ofn.lpstrFilter = szFilter[0];
ofn.lpstrCustomFilter = NULL;
ofn.nMaxCustFilter = 0L;
ofn.nFilterIndex = 1L;
ofn.lpstrFile = szFile;
ofn.nMaxFile = sizeof(szFile);
ofn.lpstrFileTitle = szFileTitle;
ofn.nMaxFileTitle = sizeof(szFileTitle);
ofn.lpstrInitialDir = szDirName;
ofn.lpstrTitle = NULL;
ofn.Flags = OFN_ENABLEHOOK | OFN_ENABLETEMPLATE;
ofn.nFileOffset = 0;
ofn.nFileExtension = 0;
ofn.lpstrDefExt = NULL;
ofn.lpfnHook = MakeProcInstance((FARPROC) FileOpenHookProc,
    hInst);
ofn.lpTemplateName = "FileOpen";
```

In diesem Beispiel wird die Funktion *MakeProcInstance* aufgerufen, um eine Adresse für die Prozedurinstanz der Filterfunktion zu erzeugen. Diese Adresse wird dem Element *lpfnHook* der *OPENFILENAME*-Struktur zugewiesen. Wenn die Filterfunktion Teil einer dynamischen Linkbibliothek und nicht einer Anwendung ist, wird die Funktionsadresse durch den Aufruf der Funktion *GetProcAddress* anstelle von *MakeProcInstance* ermittelt.

Die Filterfunktion bearbeitet alle Botschaften oder Benachrichtigungen, die das anwendungsdefinierte Dialogfenster benötigt. Mit Ausnahme der Botschaft *WM_INITDIALOG* erhält die Filterfunktion Botschaften oder Benachrichtigungen vor der in COMMDLG.DLL bereitgestellten Dialogfensterfunktion. Im Falle von *WM_INITDIALOG* erhält die Filterfunktion die Botschaft nach der Dialogfensterfunktion und sollte sie bearbeiten. Wenn die Filterfunktion die Bearbeitung einer Botschaft beendet hat, gibt sie einen Wert zurück, der anzeigt, ob die in COMMDLG.DLL bereitgestellte Dialogfensterfunktion die Botschaft auch bearbeiten soll. Wenn die Dialogfensterfunktion die Botschaft bearbeiten soll, ist der Rückgabewert *FALSE*. Soll die Dialogfensterfunktion die Botschaft ignorieren, ist der Rückgabewert *TRUE*.

Um die Botschaft des *OK*-Schalters zu bearbeiten, nachdem die Dialogfensterfunktion sie bearbeitet hat, muß eine Anwendung eine Botschaft an sich selbst senden, sobald sie die *OK*-Botschaft empfängt. Wenn die Anwendung die selbst gesendete Botschaft erhält, hat die Standarddialogfensterfunktion die Bearbeitung der Botschaften für das Dialogfenster beendet. Diese Technik ist insbesondere bei der Arbeit mit den Dialogfenstern *Suchen* und *Ersetzen* hilfreich, weil das Element *Flags* der *FINDREPLACE*-Struktur Änderungen des Dialogfensters erst widerspiegelt, wenn die Botschaften von COMMDLG.DLL bearbeitet worden sind.

Das folgende Beispiel zeigt eine Filterfunktion für ein anwendungsdefiniertes *Öffnen*-Dialogfenster:

```
UINT CALLBACK FileOpenHookProc(HWND hdlg, UINT msg,
    WPARAM wParam, LPARAM lParam)
{
    switch (msg) {
        case WM_INITDIALOG:
            return TRUE;

        case WM_COMMAND:

            /* lCustData mit Hilfe der Funktion
               IsDlgButtonChecked setzen. */

            if (wParam == IDOK) {

                /* Backup-Flag setzen. */

                ofn.lCustData =
                  (DWORD) IsDlgButtonChecked(hdlg, ID_CUSTCHX);
            }
            return FALSE; /* Standardbearbeitung ermöglichen. */
    }

    /* Standardbearbeitung ermöglichen. */
    return FALSE;
}
```

Diese Filterfunktion testet ein anwendungsdefiniertes Auswahlfeld, wenn der Benutzer den *OK*-Schalter wählt. War das Auswahlfeld markiert, setzt die Filterfunktion das Element *lCustData* der *OPENFILENAME*-Struktur auf 1, andernfalls auf 0.

Enthält die Filterfunktion Befehle, die einen außergewöhnlichen Abbruch des Standarddialogfensters durchführen, sollte dieser Quelltext stattdessen den Wert *IDABORT* mit Hilfe der Funktion *PostMessage* an die Dialogfensterfunktion übergeben:

```
PostMessage(hDlg, WM_COMMAND, IDABORT, (LONG) FALSE);
```

Wenn eine Filterfunktion den Wert *IDABORT* sendet, liefert die Standarddialogfensterfunktion den im niederwertigen *WORD* des Parameters *lParam* enthaltenen Wert zurück. Wenn z.B. die Filterfunktion für *GetOpenFileName* die Funktion *PostMessage* mit (LONG) 100 als letztem Parameter aufgerufen hat, würde *GetOpenFileName* 100 zurückgeben.

Eine Filterfunktion muß in der Moduldefinitionsdatei (.DEF) einer Anwendung exportiert werden:

```
NAME cd

EXETYPE WINDOWS

STUB    'WINSTUB.EXE'

CODE    PRELOAD MOVEABLE DISCARDABLED

DATA    PRELOAD MOVEABLE MULTIPLE

HEAPSIZE  1024
```

```
STACKSIZE 8192

EXPORTS
  FILEOPENHOOKPROC @  1
```

Eine Dialogfensterschablone anpassen

Die in COMMDLG.DLL bereitgestellten Dialogfensterschablonen für Standarddialogfenster enthalten die Daten, die die Dialogfensterfunktion zur Anzeige des Dialogfensters verwendet. Die meisten Anwendungen, die ein Standarddialogfenster anpassen, müssen auch eine selbstdefinierte Dialogfensterschablone erzeugen, die anstelle der Dialogfensterschablone in COMMDLG.DLL verwendet wird. Eine selbstdefinierte Dialogfensterschablone wird nicht für alle anwendungsdefinierten Dialogfenster benötigt. Eine Schablone würde z.B. nicht benötigt werden, wenn eine Anwendung ein Dialogfenster nur relativ geringfügig und für ungewöhnliche Situationen geändert hat.

Ein Entwickler sollte eine selbstdefinierte Dialogfensterschablone erzeugen, indem er die entsprechende Dialogfensterschablone in COMMDLG.DLL ändert. Die folgende Aufstellung zeigt die Dateinamen der Schablonen und die Namen der entsprechenden Standarddialogfenster:

Schablonendateiname	Entsprechendes Dialogfenster
COLOR.DLG	*Farben*
FILEOPEN.DLG	*Datei öffnen* (Einfachauswahl)
FILEOPEN.DLG	*Datei öffnen* (Mehrfachauswahl)
FINDTEXT.DLG	*Suchen*
FINDTEXT.DLG	*Ersetzen*
FONT.DLG	*Schriftart*
PRNSETUP.DLG	*Drucken*
PRNSETUP.DLG	*Druckereinrichtung*

Der folgende Quelltextauszug stammt aus einer für ein *Öffnen*-Dialogfenster erzeugten Dialogfensterschablone:

```
CONTROL "&Sicherungsdatei", ID_CUSTCHX, "button",
        BS_AUTOCHECKBOX | WS_CHILD | WS_TABSTOP | WS_GROUP,
        208, 86, 50, 12
END
```

Dieser Eintrag fügt ein neues Auswahlfeld *Sicherungsdatei* unmittelbar unterhalb des bereits vorhandenen Felds *Schreibgeschützt* in das Dialogfenster ein.

Die selbstdefinierte Schablone sollte zur Ressourcendatei der Anwendung hinzugefügt werden.

Das selbstdefinierte Dialogfenster anzeigen

Nachdem Ihre Anwendung die Filterfunktion und die Dialogfensterschablone erzeugt hat, sollte sie die Elemente der Struktur des Standarddialogfensters

entsprechend setzen und die geeignete Funktion aufrufen, um das selbstdefinierte Dialogfenster anzuzeigen.

Das folgende Beispiel ruft die Funktion *GetOpenFileName* auf und erzeugt eine Sicherungsdatei, wenn der Benutzer das Auswahlfeld *Sicherungsdatei* im angepaßten Dialogfenster gewählt hat:

```
/* Datei öffnen und Sicherungskopie erzeugen. */
if (GetOpenFileName(&ofn)) {
    hf = _lopen(ofn.lpstrFile, OF_READWRITE);
    /* Sicherungsdatei erzeugen. */
    if (ofn.lCustData) {
        /* Dateien mit Namenserweiterung bearbeiten. */
        if (ofn.FileExtension) {
            for (i=0; i<(int)ofn.nFileExtension; i++)
                szChar[i] = *ofn.lpstrFile++;
        } /* endif */
        /* Dateien ohne Namenserweiterung bearbeiten. */
        else {
            i = 0;
            while (*ofn.lpstrFile != '\0')
                szChar[i++] = *ofn.lpstrFile++;
            szChar[i] = '.';
        } /* end else */
        pszNewPAFN = lstrcat(szChar, "BAK");
        /* Sicherungsdatei erzeugen. */
        hfBackup = _lcreat(pszNewPAFN, 0);
        /* Inhalt der Originaldatei in die Sicherungsdatei
           kopieren. */
        while ((cBufLngth = _lread(hf, cBuf1, 256)) == 256)
            _lwrite(hfBackup, cBuf1, cBufLngth);
        _lwrite(hfBackup, cBuf1, cBufLngth);
        _lclose(hfBackup);
    } /* endif GetOpenFileName */
    /* Hier beginnen die Dateioperationen. */
} /* endif (GetOpenFileName) */
```

Die folgende Abbildung zeigt das selbstdefinierte Dialogfenster *Datei öffnen*. Das neue Auswahlfeld *Sicherungsdatei* erscheint unterhalb der Standardschalter.

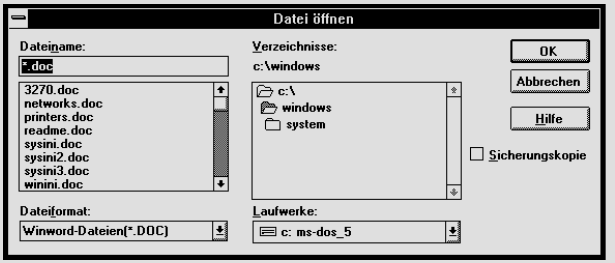


Abbildung 1.11 Das Dialogfenster Datei öffnen

Hilfe für das Standarddialogfenster bereitstellen

Eine Anwendung kann einen *Hilfe*-Schalter in einem Standarddialogfenster anzeigen, indem sie das entsprechende Flag im Element *Flags* der Struktur für das Standarddialogfenster setzt.

Die folgende Liste zeigt die Strukturen für die Standarddialogfenster und das entsprechende *Hilfe*-Flag:

Struktur	Flag
OPENFILENAME	*OFN_SHOWHELP*
CHOOSECOLOR	*CC_SHOWHELP*
FINDREPLACE	*FR_SHOWHELP*
CHOOSEFONT	*CF_SHOWHELP*
PRINTDLG	*PD_SHOWHELP*

Wenn eine Anwendung den *Hilfe*-Schalter anzeigt, muß sie die entsprechenden Anforderungen des Benutzers bearbeiten. Dies kann entweder in einer der Fensterfunktionen der Anwendung oder in einer Filterfunktion geschehen.

Wenn die Anwendung die Anforderung in einer Fensterfunktion bearbeitet, muß sie zuerst einen neuen Botschaftsbezeichner für den von der Konstanten *HELPMSGSTRING* definierten String erzeugen, indem sie die Funktion *RegisterWindowMessage* aufruft und diese Konstante als einzigen Parameter übergibt. Weitere Informationen über die Verarbeitung registrierter Fensterbotschaften finden Sie im Abschnitt »Die Dialogfenster *Suchen* und *Ersetzen*«.

Außer der Erzeugung des neuen Botschaftsbezeichners muß die Anwendung das Element *hwndOwner* der Datenstruktur des Dialogfensters mit dem Handle des besitzenden Fensters initialisieren. Nachdem der Botschaftsbezeichner erzeugt und das Element *hwndOwner* gesetzt ist, benachrichtigt die Dialogfensterfunktion das besitzende Fenster, sobald der Benutzer der Hilfe-Schalter auswählt.

Das folgende Beispiel bearbeitet die Hilfe-Anforderung des Benutzers in der Fensterfunktion des besitzenden Fensters. Die **if**-Anweisung sollte sich im Abschnitt **default:** der **switch**-Anweisung befinden.

```
MyHelpMsg = RegisterWindowMessage(HELPMSGSTRING);
          .
          .
          .
if (message == MyHelpMsg)
   WinHelp(hWnd, "appfile.hlp", HELP_CONTEXT, ID_MY_CONTEXT);
```

Wenn die Anwendung die Hilfe-Anforderung in einer Filterfunktion bearbeitet, sollte sie folgende Bedingung in der *WM_COMMAND*-Botschaft überprüfen:

```
wParam == pshHelp
```

Ist diese Bedingung erfüllt, sollte die Filterfunktion die Funktion *WinHelp* – wie im obigen Beispiel gezeigt – aufrufen. Um Hilfe-Anforderungen in einer Filterfunktion zu bearbeiten, muß die Header-Datei dlsg.h aufgenommen werden. Diese enthält den Quelltext der Filterfunktion.

Fehler-überwachung

Schlägt eine Standarddialogfensterfunktion fehl, können Anwendungen die Funktion *CommDlgExtendedError* aufrufen, um den Grund für den Fehler zu ermitteln. Die Funktion gibt einen Fehlerwert zurück, der den Grund des zuletzt aufgetretenen Fehlers bezeichnet.

In der Header-Datei cderr.h sind sechs Konstanten definiert, die Fehlerwertbereiche für bestimmte Fehlerarten bezeichnen. Die Fehlerwerte werden von der Funktion *CommDlgExtendedError* zurückgegeben. In der folgenden Tabelle sind die Konstanten in steigender Reihenfolge nach ihren Fehlerwerten aufgeführt.

Konstante	Bedeutung
CDERR_GENERAL-CODES	Allgemeine Fehlercodes für Standarddialogfenster. Diese Fehler bewegen sich zwischen den Werten 0x000 und 0x0FFF.
PDERR_PRINTER-CODES	Fehlercodes für das Dialogfenster *Drucken*. Der Wertebereich liegt zwischen 0x1000 und 0x1FFF.
CFERR_CHOOSEFONT-CODES	Fehlercodes für das Dialogfenster *Schriftart*. Der Wertebereich liegt zwischen 0x2000 und 0x2FFF.
FNERR_FILENAME-CODES	Fehlerwerte für die Dialogfenster *Datei öffnen* und *Datei speichern unter*. Der Wertebereich liegt zwischen 0x3000 und 0x3FFF.
FRERR_FINDREPLACE-CODES	Fehlerwerte für die Dialogfenster *Suchen* und *Ersetzen*. Der Wertebereich liegt zwischen 0x4000 und 0x4FFF.
CCERR_CHOOSECOLOR-CODES	Fehlerwerte für das Dialogfenster *Farben*. Der Wertebereich liegt zwischen 0x5000 und 0x5FFF.

2. Die DDE-Bibliothek (DDEML)

Dieses Kapitel beschreibt, wie die Bibliothek zur Verwaltung des dynamischen Datenaustauschs (DDEML) verwendet wird. Die DDEML ist eine dynamische Linkbibliothek (DLL), durch die Windows-Anwendungen Daten gemeinsam benutzen können. Die folgenden Themen sind mit den Informationen in diesem Kapitel verwandt:

- Atome
- Speicherverwaltung
- Zwischenablage
- Dynamische Linkbibliotheken
- OLE (Object linking and embedding)

Der dynamische Datenaustausch (DDE) ist eine Form der Kommunikation zwischen Prozessen, der gemeinsam benutzten Speicher zum Austausch von Daten zwischen Anwendungen verwendet. Anwendungen können DDE für einzelne Datentransfers und für laufende Aktualisierungen verwenden, bei denen die Anwendungen Daten austauschen, sobald neue verfügbar sind.

Der dynamische Datenaustausch unterscheidet sich vom Datentransfermechanismus über die Zwischenablage, der ebenfalls ein Teil von Windows ist. Ein Unterschied besteht darin, daß die Zwischenablage fast immer als eine einmalige Antwort auf eine bestimmte Aktion eines Benutzers verwendet wird – wie z.B. die Auswahl des Befehls *Einfügen* aus einem Menü. Obwohl DDE auch von einem Benutzer initiiert werden kann, wird er normalerweise ohne weitere Beteiligung des Benutzers fortgesetzt.

Die DDEML stellt eine Anwendungsprogrammierschnittstelle (API) bereit, die das Hinzufügen von DDE-Fähigkeiten zu einer Windows-Anwendung vereinfacht. Anstatt DDE-Botschaften direkt zu senden, zu übermitteln und zu bearbeiten, verwaltet eine Anwendung DDE-Konversationen mit Hilfe der DDEML-Funktionen. Eine DDE-Konversation ist die Interaktion zwischen Client- und Server-Anwendungen. Die DDEML bietet außerdem Hilfen zur Verwaltung der von DDE-Anwendungen gemeinsam benutzten Strings und Daten. Anstatt Atome und Zeiger auf gemeinsam benutzte Speicherobjekte zu verwenden, erzeugen DDE-Anwendungen String-Handles, die Strings bezeichnen und Daten-Handles, die globale Speicherobjekte bezeichnen. Diese Handles werden dann ausgetauscht. Die DDEML ermöglicht Server-Anwendungen die Registrierung der von ihr unterstützten Servicenamen. Die Namen werden den

anderen Anwendungen im System mitgeteilt, die diese dann zur Verbindungsaufnahme mit dem Server verwenden können. Die DDEML sorgt auch für Kompatibiltät zwischen DDE-Anwendungen, indem sie diese veranlaßt, das DDE-Protokoll in einer konsistenten Art und Weise zu implementieren.

Bestehende Anwendungen, die das auf Botschaften basierende DDE-Protokoll verwenden, sind vollständig kompatibel zu denen, die die DDEML verwenden. Botschaftsbasierende DDE-Anwendungen können also mit Anwendungen Konversationen einrichten und Transaktionen austauschen, die die DDEML verwenden. Wegen der vielen Vorteile der DDEML sollte sie von neuen Anwendungen anstelle der DDE-Botschaften verwendet werden.

Die DDEML kann auf Systemen laufen, auf denen Windows 3.0 oder eine spätere Version installiert ist. Sie unterstützt jedoch nicht den Real-Modus. Um die API-Elemente der DDEML zu verwenden, müssen Sie die Header-Datei ddeml.h in Ihre Quelltextdatei einschließen, sie mit DDEML.LIB linken und dafür sorgen, daß DDEML.DLL sich im Pfad Ihres Systems befindet.

Grundlagen

Die Grundlagen in diesem Abschnitt sind der Schlüssel zum Verständnis des DDE-Protokolls und der DDEML-Bibliothek.

Interaktion von Client und Server

Dynamischer Datenaustausch findet immer zwischen einer Client- und einer Server-Anwendung statt. Der Client leitet den Austausch ein, indem er eine Konversation mit dem Server eröffnet, damit er Transaktionen an den Server senden kann. Eine Transaktion ist die Anforderung von Daten oder Diensten. Der Server antwortet auf diese Transaktionen, indem er dem Client die Daten oder Dienste bereitstellt. Ein Server kann mehrere Clients zur selben Zeit haben und ein Client kann von mehreren Server-Anwendungen Daten anfordern. Eine Anwendung kann sowohl Server als auch Client sein. Ein Client beendet eine Konversation, wenn er die Daten oder Dienste eines Server nicht mehr benötigt. Eine Grafikanwendung könnte z.B. eine Balkengrafik enthalten, die die monatlichen Gewinne einer Firma anzeigt. Die Daten für die Balkengrafik könnten in einer Tabellenkalkulation enthalten sein. Um die neueste Gewinnabbildung anzufordern, eröffnet die Grafikanwendung (der Client) eine Konversation mit der Tabellenkalkulation (dem Server). Die Grafikanwendung sendet dann eine Transaktion an die Tabellenkalkulation und fordert neue Gewinndaten an.

Transaktionen und die DDE-Callback-Funktion

Die DDEML benachrichtigt eine Anwendung von DDE-Aktivitäten, die diese beeinflussen, indem sie Transaktionen an die DDE-Callback-Funktion der An-

wendung sendet. Eine Transaktion ähnelt einer Botschaft. Sie ist eine benannte Konstante, die von anderen Parametern begleitet wird, die zusätzliche Informationen über die Transaktion enthalten.

Die DDEML übergibt eine Transaktion an eine anwendungsdefinierte DDE-Callback-Funktion, die eine dem Typ der Transaktion entsprechende Aktion durchführt. Wenn z.B. eine Client-Anwendung versucht, eine Konversation mit einer Server-Anwendung zu eröffnen, ruft der Client die Funktion *DdeConnect* auf. Dies veranlaßt die DDEML, eine *XTYP_CONNECT*-Transaktion an die DDE-Callback-Funktion des Server zu senden. Diese kann die Konversation zulassen, indem sie *TRUE* an die DDEML zurückliefert oder sie kann die Konversation ablehnen, indem sie *FALSE* zurückgibt. Eine detaillierte Beschreibung der Transaktionen finden Sie im Abschnitt »Transaktionsverwaltung«.

Servicenamen, Themennamen und Elementnamen

Ein DDE-Server verwendet eine dreistufige Hierarchie – Servicename (Anwendungsname in der früheren DDE-Dokumentation), Themenname und Elementname –, um Daten eindeutig zu bezeichnen, die der Server während einer Konversation austauschen kann. Ein Servicename ist ein String, auf den eine Server-Anwendung antwortet, wenn ein Client eine Konversation mit dem Server einrichten will. Ein Client muß diesen Servicenamen angeben, um die Konversation mit dem Server einrichten zu können. Ein Server kann zwar grundsätzlich auf mehrere Servicenamen antworten, reagiert aber meist nur auf einen Namen.

Ein Themenname ist ein String, der einen virtuellen Datenkontext bezeichnet. Bei Server-Anwendungen, die in Dateien organisierte Dokumente bearbeiten, sind Themennamen normalerweise Dateinamen; bei anderen Server-Anwendungen sind sie andere anwendungsspezifische Strings. Ein Client muß einen Themennamen zusammen mit einem Server-Servicenamen angeben, wenn er versucht, eine Konversation mit einem Server einzurichten.

Ein Elementname ist ein String, der eine Dateneinheit bezeichnet, die ein Server während einer Transaktion an einen Client übergeben kann. So könnte z.B. ein Elementname einen Integer, einen String, mehrere Absätze eines Texts oder ein Bitmap bezeichnen.

Für einen Client sind diese Namen die Schlüssel, um eine Konversation mit einem Server einzurichten und Daten von ihm zu erhalten.

Das Thema System

Das Thema *System* stellt einen Kontext für Informationen bereit, die für jeden DDE-Client von allgemeinem Interesse sein könnten. Server-Anwendungen sollten das Thema *System* immer unterstützen. Es ist in der DDEML-Header-Datei als *SZDDESYS_TOPIC* definiert.

Um herauszufinden, welche Server vorhanden sind und welche Arten von

Informationen sie liefern können, kann eine Client-Anwendung beim Start eine Konversation über das Thema *System* mit auf *NULL* gesetztem Servicenamen anfordern. Diese Art Konversation sollte möglichst selten verwendet werden, da sie viel Systemleistung in Anspruch nimmt.

Weitere Informationen über die Einleitung von DDE-Konversationen finden Sie im Abschnitt »Konversationsverwaltung«.

Ein Server sollte die folgenden Elementnamen innerhalb des Themas *System* sowie alle anderen Elementnamen unterstützen, die für einen Client nützlich sein könnten:

Element	Beschreibung
SZDDE_ITEM_ITEMLIST	Eine Liste der Elemente, die unter einem Nicht-*System*-Thema unterstützt werden. Die Liste kann sich in Abhängigkeit vom Zeitpunkt und vom Thema ändern.
SZDDESYS_ITEM_FORMATS	Eine Liste der Nummern der Zwischenablageformate, die der Server übertragen kann. Diese Liste sollte so geordnet sein, daß die aussagekräftigsten Formate zuerst erscheinen. Ein Server ist möglicherweise nicht in der Lage, alle Elemente in allen Formaten dieser Liste zu übertragen. Er sollte aber zumindest das *CF_TEXT*-Zwischenablageformat für Elementnamen unterstützen, die dem Thema *System* zugeordnet sind.
SZDDESYS_ITEM_HELP	Allgemeine Hilfe-Informationen.
SZDDESYS_ITEM_RTNMSG	Stellt Details über die zuletzt verwendete *WM_DDE_ACK*-Botschaft bereit. Dies ist nützlich, wenn mehr als 8 Bits anwendungsspezifischer Rückgabedaten benötigt werden.
SZDDESYS_ITEM_STATUS	Anzeige des aktuellen Status des Server. Normalerweise unterstützt dieses Element nur das *CF_TEXT*-Format und enthält den *Ready*- oder *Busy*-String.
SZDDESYS_ITEM_SYSITEMS	Eine Liste der Elemente, die von diesem Server unter dem Thema *System* unterstützt werden.
SZDDESYS_ITEM_TOPICS	Eine Liste der von diesem Server aktuell unterstützten Themen. Diese Liste kann sich jederzeit ändern.

Diese Elementnamen sind String-Konstanten und in den DDEML-Header-Dateien definiert. Um String-Handles für diese Strings zu erhalten, muß eine Anwendung die DDEML-Funktionen zur String-Verwaltung verwenden, wie sie es auch für jeden anderen String in einer DDEML-Anwendung tun würde. Weitere Informationen über die Verwaltung von Strings können Sie dem Abschnitt »String-Verwaltung« entnehmen.

Initialisierung

Die dynamische Linkbibliothek DDEML läuft nur unter Windows. Vor dem Aufruf einer DDEML-Funktion sollte eine Anwendung die Funktion *GetWinFlags* aufrufen und den Rückgabewert auf das Flag *WF_PMODE* überprüfen. Wenn dieses Flag zurückgeben wird, kann die Anwendung DDEML-Funktionen aufrufen.

Vor dem Aufruf einer anderen DDEML-Funktion muß eine Anwendung die Funktion *DdeInitialize* aufrufen. Diese ermittelt einen Instanzenbezeichner für die Anwendung, registriert deren DDE-Callback-Funktion für die DDEML und gibt die Filter-Flags für Transaktionen für die Callback-Funktion an.

Die DDEML verwendet Instanzenbezeichner, damit sie Anwendungen unterstützen kann, die mehrere DDEML-Instanzen zulassen. Jede Instanz einer Anwendung muß ihren Instanzenbezeichner als Parameter *idInst* an jede andere DDEML-Funktion übergeben, die ihn benötigt. Eine Anwendung, die mehrere DDEML-Instanzen verwendet, sollte jeder Instanz eine andere DDE-Callback-Funktion zuweisen. Dadurch kann die Anwendung jede Instanz innerhalb ihrer Callback-Funktion bezeichnen.

Der Zweck mehrfacher DDEML-Instanzen ist die Unterstützung dynamischer Linkbibliotheken, die DDEML benutzen. Es ist nicht ratsam, daß eine Anwendung mehrere DDE-Instanzen besitzt.

Transaktionsfilter optimieren die Leistung des Systems, indem sie verhindern, daß die DDEML ungewollte Transaktionen an die DDE-Callback-Funktion der Anwendung übergibt. Eine Anwendung legt die Transaktionsfilter beim Aufruf der Funktion *DdeInitialize* fest. Sie sollte für jeden Transaktionstyp, den sie in ihrer Callback-Funktion nicht bearbeitet, ein Transaktionsfilter-Flag setzen. Die Transaktionsfilter können durch nachfolgende Aufrufe der Funktion *DdeInitialize* geändert werden. Eine vollständige Liste der Transaktionsfilter-Flags können Sie der Beschreibung der Funktion *DdeInitialize* entnehmen.

Weitere Informationen über Transaktionen finden Sie im Abschnitt »Transaktionsverwaltung«.

Das folgende Beispiel zeigt, wie eine Anwendung initialisiert werden muß, damit sie die DDEML verwenden kann:

```
DWORD idInst = 0L;          /* Instanzenbezeichner */
HANDLE hInst;               /* Handle der Instanz */
FARPROC lpDdeProc;          /* Adresse der Prodezurinstanz */

lpDdeProc = MakeProcInstance((FARPROC) DdeCallback, hInst);

if (DdeInitialize(&idInst,   /* empfängt den Instanzenbezeichner */
    (PFNCALLBACK) lpDdeProc, /* Adresse der Callback-Funktion */
    CBF_FAIL_EXECUTES |      /* Filter für XTYP_EXECUTE-
                                Transaktionen */
    CBF_FAIL_POKES, 0L);     /* Filter für XTYP_POKE-Transaktionen */
    return FALSE;
```

Dieses Beispiel ermittelt eine Adresse für die Prozedurinstanz der Callback-Funktion *DdeCallback* und übergibt dann die Adresse an die DDEML. Die Filter *CBF_FAIL_EXECUTES* und *CBF_FAIL_POKES* verhindern, daß die DDEML

XTYP_EXECUTE- oder *XTYP_POKE*-Transaktionen an die Callback-Funktion übergibt.

Eine Anwendung sollte die Funktion *DdeUninitialize* aufrufen, wenn sie die DDEML nicht länger benötigt. Diese Funktion beendet jede aktuell für die Anwendung geöffnete Konversation und gibt die DDEML-Ressourcen frei, die das System für die Anwendung reserviert hat.

Die DDEML kann beim Beenden einer Konversation Schwierigkeiten haben. Dies geschieht, wenn der Partner in einer Konversation diese nicht korrekt beendet. In diesem Fall wartet das System in einer modalen Schleife auf das Beenden aller Konversationen. Dieser Schleife ist ein systemdefinierter Timeout-Wert zugeordnet. Wenn die Timeout-Periode vor Beendigung der Konversationen abläuft, erscheint ein Meldungsfenster, das dem Anwender drei Auswahlmöglichkeiten gibt: eine weitere Timeout-Periode zu warten (Wiederholen), auf unbestimmte Zeit zu warten (Ignorieren) oder die modale Schleife zu verlassen (Abbrechen). Eine Anwendung sollte *DdeUninitialize* aufrufen, nachdem sie für den Anwender unsichtbar geworden und ihre Botschaftsschleife beendet ist.

Callback-Funktion

Eine Anwendung, die die DDEML verwendet, muß eine Callback-Funktion bereitstellen, um die sie betreffenden DDE-Ereignisse zu bearbeiten. Die DDEML benachrichtigt eine Anwendung von solchen Ereignissen, indem sie Transaktionen an deren DDE-Callback-Funktion sendet. Die Transaktionen, die eine Callback-Funktion erhält, sind von den Filter-Flags, die die Anwendung in der *DdeInitialize*-Funktion gesetzt hat und davon abhängig, ob die Anwendung ein Server, ein Client oder beides ist. Das folgende Beispiel zeigt die allgemeine Struktur einer Callback-Funktion für eine typische Client-Anwendung:

```
HDDEDATA EXPENTRY DdeCallback(wType, wFmt, hConv, hsz1,
   hsz2, hData, dwData1, dwData2)
WORD wType;      /* Transaktionstyp                      */
WORD wFmt;       /* Zwischenablageformat                 */
HCONV hConv;     /* Handle der Konversation              */
HSZ hsz1;        /* Handle eines String                  */
HSZ hsz2;        /* Handle eines String                  */
HDDEDATA hData;  /* Handle eines globalen Speicherobjekts */
DWORD dwData1;   /* transaktionsspezifische Daten        */
DWORD dwData2;   /* transaktionsspezifische Daten        */
{
    switch (wType) {
        case XTYP_REGISTER:
        case XTYP_UNREGISTER:
            .
            .
            .
```

```
            return (HDDEDATA) NULL;
        case XTYP_ADVDATA:
            .
            .
            .
            return (HDDEDATA) DDE_FACK;
        case XTYP_XACT_COMPLETE:
            .
            .
            .
            return (HDDEDATA) NULL;
        case XTYP_DISCONNECT:
            .
            .
            .
            return (HDDEDATA) NULL;
        default:
            return (HDDEDATA) NULL;
    }
}
```

Der Parameter *wType* gibt den von der DDEML an die Callback-Funktion gesendeten Transaktionstyp an. Die Werte der restlichen Parameter sind von diesem Typ abhängig. Die Transaktionstypen und die sie erzeugenden Ereignisse sind in den folgenden Abschnitten dieses Kapitels beschrieben. Detaillierte Informationen über alle Transaktionstypen finden Sie im Abschnitt »Transaktionsverwaltung«.

String-Verwaltung

Viele DDEML-Funktionen müssen auf Strings zugreifen, um eine DDE-Aufgabe durchzuführen. Ein Client muß z.B. beim Aufruf der Funktion *DdeConnect* einen Service- und einen Themennamen angeben, um eine Konversation mit einem Server einzurichten. Eine Anwendung gibt einen String an, indem sie ein String-Handle anstelle eines Zeigers in einer DDEML-Funktion übergibt. Ein String-Handle ist ein vom System zugewiesener *DWORD*-Wert, der einen String bezeichnet.

Eine Anwendung kann ein String-Handle für einen bestimmten String durch die Funktion *DdeCreateStringHandle* anfordern. Diese Funktion registriert den String für das System und gibt ein String-Handle an die Anwendung zurück. Die Anwendung kann das Handle an DDEML-Funktionen übergeben, die auf den String zugreifen müssen.

Das folgende Beispiel fordert String-Handles für die Strings des Themas *System* und den Servicenamen an.

```
HSZ hszServName;
HSZ hszSysTopic;

hszServName = DdeCreateStringHandle(
    idInst,         /* Instanzenbezeichner       */
    "MyServer",     /* zu registrierender String */
    CP_WINANSI);    /* Code-Seite                */

hszSysTopic = DdeCreateStringHandle(
    idInst,         /* Instanzenbezeichner       */
    SZDDESYS_TOPIC, /* Thema System              */
    CP_WINANSI);    /* Code-Seite                */
```

Der Parameter *idInst* gibt in diesem Beispiel den Instanzenbezeichner an, der durch die Funktion *DdeInitialize* ermittelt wurde.

Die DDE-Callback-Funktion einer Anwendung empfängt ein oder mehrere String-Handles während der meisten DDE-Transaktionen. Ein Server erhält z.B. zwei String-Handles während der Transaktion *XTYP_REQUEST*: einer bezeichnet einen String, der einen Themennamen angibt; der andere bezeichnet einen String, der einen Elementnamen angibt. Eine Anwendung kann die Länge des String ermitteln, der einem String-Handle entspricht und den String in einen anwendungsdefinierten Puffer kopieren, indem sie die Funktion *DdeQueryString* aufruft:

```
DWORD idInst;
DWORD cb;
HSZ hszServ;
PSTR pszServName;

cb=DdeQueryString(idInst, hszServ, (LPSTR) NULL, 0L, (WINANSI)+1;
pszServName = (PSTR) LocalAlloc(LPTR, (WORD) cb);
DdeQueryString(idInst, hszServ, pszServName, cb, CP_WINANSI);
```

Ein instanzenspezifisches String-Handle kann nicht auf einen String und dieser dann wieder auf das String-Handle abgebildet werden. Im folgenden Beispiel erzeugt z.B. die Funktion *DdeQueryString* einen String aus einem String-Handle und *DdeCreateStringHandle* erzeugt dann aus diesem String ein String-Handle. Die beiden Handles sind jedoch nicht identisch:

```
DWORD cb;
HSZ hszInst, hszNew;
PSZ pszInst;

DdeQueryString(idInst, hszInst, pszInst, cb, CP_WINANSI);
hszNew = DdeCreateStringHandle(idInst, pszInst, CP_WINANSI);
/* hszNew != hszInst */
```

Ein String-Handle, das an die DDE-Callback-Funktion einer Anwendung übergeben wird, wird ungültig, wenn die Funktion zurückkehrt. Eine Anwendung kann mit der Funktion *DdeKeepStringHandle* ein String-Handle speichern, um es nach Rückkehr der Callback-Funktion zu verwenden.

Wenn eine Anwendung die Funktion *DdeCreateStringHandle* aufruft, nimmt das System den angegebenen String in eine systemweite String-Tabelle auf und

erzeugt ein Handle, mit dem es auf den String zugreifen kann. Das System verwaltet auch einen Verwendungszähler für jeden String in der Tabelle.

Wenn eine Anwendung die Funktion *DdeCreateStringHandle* aufruft und einen bereits in der Tabelle enthaltenen String angibt, erhöht das System den Verwendungszähler anstatt den String ein weiteres Mal hinzuzufügen. Eine Anwendung kann diesen Zähler auch mit der Funktion *DdeKeepStringHandle* erhöhen. Wenn eine Anwendung die Funktion *DdeFreeStringHandle* aufruft, wird der Verwendungszähler verringert.

Ein String wird aus der Tabelle entfernt, sobald sein Zähler Null ist. Weil mehr als eine Anwendung das Handle eines bestimmten String anfordern kann, sollte eine Anwendung ein String-Handle nicht öfter freigeben, als sie es erzeugt oder gespeichert hat. Andernfalls könnte der String aus der Tabelle entfernt und anderen Anwendungen der Zugriff auf den String verweigert werden.

Die Funktionen der DDEML-String-Verwaltung basieren auf der Atomverwaltung von Windows und unterliegen den gleichen Größenbeschränkungen wie Atome.

Service-namen

Die DDEML ermöglicht es einer Server-Anwendung, die von ihr unterstützten Servicenamen zu registrieren. Dadurch sendet die DDEML keine *XTYP_CONNECT*-Transaktionen für nicht unterstützte Servicenamen an die DDE-Callback-Funktion des Server.

Servicenamen registrieren

Durch die Registrierung seiner Servicenamen bei der DDEML informiert ein Server andere DDE-Anwendungen im System, daß ein neuer Server zur Verfügung steht. Ein Server registriert einen Servicenamen, indem er die Funktion *DdeNameService* aufruft und ein String-Handle angibt, das den Servicenamen bezeichnet. Anschließend sendet die DDEML eine *XTYP_REGISTER*-Transaktion an die Callback-Funktion jeder DDEML-Anwendung im System (außer an diejenigen, die das Filter-Flag *CBF_SKIP_REGISTRATIONS* in der Funktion *DdeInitialize* angegeben haben). Die *XTYP_REGISTER*-Transaktion übergibt zwei String-Handles an eine Callback-Funktion: Das erste bezeichnet den String, der den eigentlichen Servicenamen angibt, das zweite bezeichnet den String, der den instanzenspezifischen Servicenamen angibt. Ein Client verwendet den eigentlichen Servicenamen normalerweise in einer Liste der verfügbaren Server, damit der Benutzer einen Server aus der Liste auswählen kann. Der Client benutzt den instanzenspezifischen Servicenamen, um eine Konversation mit einer bestimmten Instanz einer Server-Anwendung einzurichten, wenn mehr als eine Instanz existiert.

Ein Server kann die Registrierung eines Servicenamens durch die Funktion *DdeNameService* rückgängig machen. Dies veranlaßt die DDEML *XTYP_UNREGISTER*-Transaktionen an die anderen DDE-Anwendungen im System zu senden, um sie darüber zu informieren, daß sie diesen Namen nicht mehr zur Einrichtung von Konversationen verwenden können.

Ein Server sollte nach *DdeInitialize* die Funktion *DdeNameService* aufrufen, um seine Servicenamen zu registrieren. Er sollte die Registrierung seiner Servicenamen vor dem Aufruf der Funktion *DdeUninitialize* rückgängig machen.

Servicenamenfilter

Außer dem Registrieren von Servicenamen ermöglicht die Funktion *DdeNameService* einem Server, seinen Servicenamenfilter ein- oder auszuschalten. Wenn ein Server seinen Servicenamenfilter ausschaltet, sendet die DDEML die Transaktion *XTYP_CONNECT* immer dann an die DDE-Callback-Funktion des Server, wenn ein Client die Funktion *DdeConnect* aufruft. Der in der Funktion angegebene Servicename wird nicht berücksichtigt. Wenn ein Server seinen Servicenamenfilter einschaltet, sendet die DDEML die Transaktion *XTYP_CONNECT* nur an den Server, wenn die Funktion *DdeConnect* einen Servicenamen angibt, den der Server in einem Aufruf der Funktion *DdeNameService* festgelegt hat.

In der Voreinstellung ist der Servicenamenfilter eingeschaltet, wenn eine Anwendung *DdeInitialize* aufruft. Dadurch sendet die DDEML keine *XTYP_CONNECT*-Transaktion an einen Server, bevor dieser die benötigten String-Handles erzeugt hat. Ein Server kann seinen Servicenamenfilter ausschalten, indem er das Flag *DNS_FILTEROFF* in einem Aufruf der Funktion *DdeNameService* setzt. Das Flag *DNS_FILTERON* schaltet den Filter ein.

Konversationsverwaltung

Eine Konversation zwischen einem Client und einem Server wird immer auf Anforderung des Client eröffnet. Wenn eine Konversation eingerichtet ist, erhält jeder Partner ein Handle, das die Konversation bezeichnet. Dieses Handle wird in DDEML-Funktionen benutzt, um Transaktionen zu senden und die Konversation zu verwalten.

Ein Client kann eine Konversation mit einem einzigen Server oder mehrere Konversationen mit einem oder mehreren Server-Anwendungen anfordern. Die folgenden Themen in diesem Abschnitt beschreiben, wie eine Anwendung eine Konversation einrichtet und erklären, wie eine Anwendung Informationen über bereits eingerichtete Konversationen anfordern kann.

Einzel-konversationen

Eine Client-Anwendung fordert eine Einzelkonversation mit einem Server an, indem sie die Funktion *DdeConnect* aufruft und die String-Handles des Servicenamens des Server und des gewünschten Themennamens angibt. Die DDEML reagiert durch das Senden einer *XTYP_CONNECT*-Transaktion an die DDE-Callback-Funktion aller Server-Anwendungen, die entweder einen Servicenamen registriert haben, der dem in der Funktion *DdeConnect* angegebenen entspricht, oder die ihre Servicenamenfilter mit Hilfe der Funktion *DdeNameService* ausgeschaltet haben. Ein Server kann auch die *XTYP_CONNECT*-Transaktionen filtern, indem er das Filter-Flag *CBF_FAIL_CONNECTIONS* in der Funktion *DdeInitialize* setzt. Während der *XTYP_CONNECT*-Transaktion übergibt die DDEML den Service- und den Themennamen an den Server, der die Namen untersuchen und *TRUE* zurückliefern sollte, wenn er das Service-/Themenname-Paar unterstützt oder *FALSE*, wenn er es nicht tut.

Wenn kein Server *TRUE* auf die *XTYP_CONNECT*-Transaktion zurückgibt, empfängt der Client von der *DdeConnect*-Funktion den Wert *NULL* und es wird keine Konversation eingerichtet. Wenn ein Server *TRUE* zurückgibt, wird eine Konversation eröffnet und der Client erhält ein Konversation-Handle: ein *DWORD*-Wert, der die Konversation bezeichnet. Der Client verwendet das Handle in nachfolgenden DDEML-Aufrufen, um Daten vom Server anzufordern. Der Server empfängt die Transaktion *XTYP_CONNECT_CONFIRM*, wenn er nicht das Filter-Flag *CBF_FAIL_CONFIRMS* gesetzt hat. Diese Transaktion übergibt das Handle der Konversation an den Server.

Das folgende Beispiel fordert eine Konversation über das Thema *System* mit einem Server an, der den Servicenamen *MyServer* erkennt. Die Parameter *hszServName* und *hszSysTopic* sind vorher erzeugte String-Handles.

```
HCONV hConv;
HWND hwndParent;
HSZ hszServName;
HSZ hszSysTopic;

hConv = DdeConnect(
    idInst,              /* Instanzenbezeichner */
    hszServName,         /* String-Handle des Servicenamens */
    hszSysTopic,         /* String-Handle des Themas System */
    (PCONVCONTEXT) NULL); /* reserviert, muß NULL sein */

if (hConv == NULL) {
    MessageBox(hwndParent, "MyServer ist nicht verfügbar.",
        (LPSTR) NULL, MB_OK);
    return FALSE;
}
```

Im vorhergehenden Beispiel veranlaßt die Funktion *DdeConnect*, daß die DDE-Callback-Funktion der *MyServer*-Anwendung eine *XTYP_CONNECT*-Transaktion empfängt.

Im folgenden Beispiel antwortet der Server auf die *XTYP_CONNECT*-Trans-

aktion, indem er das Handle des Themennamen-String, das die DDEML ihm übergeben hat, mit jedem Element im Array der Themennamen-String-Handles vergleicht, die er unterstützt. Wenn der Server eine Übereinstimmung findet, richtet er die Konversation ein.

```
#define CTOPICS 5

HSZ hsz1;           /* von der DDEML übergebenes String-Handle */
HSZ ahszTopics[CTOPICS];
                    /* Array mit den unterstützten Themen */
int i;              /* Schleifenzähler */


.
. /* Verwenden Sie die switch-Anweisung, um die Transaktionstypen
.    zu untersuchen. */
.

case XTYP_CONNECT:
    for (i = 0; i < CTOPICS; i++) {
        if (hsz1 == ahszTopics[i])
            return TRUE;  /* Konversation einrichten */
    }

    return FALSE;         /* Thema wird nicht unterstützt,
                             Konversation abgelehnt */

.
. /* Andere Transaktionstypen bearbeiten. */
.
```

Wenn der Server als Reaktion auf eine *XTYP_CONNECT*-Transaktion *TRUE* zurückliefert, sendet die DDEML eine *XTYP_CONNECT_CONFIRM*-Transaktion an seine DDE-Callback-Funktion. Der Server kann das Handle für die Konversation durch Bearbeiten dieser Transaktion erhalten.

Ein Client kann eine Platzhalter-Konversation einrichten, indem er *NULL* für das String-Handle des Servicenamens, des Themennamens oder für beide in einem Aufruf der Funktion *DdeConnect* angibt. Wenn mindestens eines der String-Handles *NULL* ist, sendet die DDEML die Transaktion *XTYP_WILDCONNECT* an die Callback-Funktionen aller DDE-Anwendungen (außer an diejenigen, die die *XTYP_WILDCONNECT*-Transaktion filtern). Jede Server-Anwendung sollte daraufhin ein Daten-Handle zurückliefern, das ein nullterminiertes Array von *HSZPAIR*-Strukturen bezeichnet. Server, die ihren Servicenamen nicht mit der Funktion *DdeNameService* registriert haben und deren Filter eingeschaltet ist, erhalten keine *XTYP_WILDCONNECT*-Transaktionen. Weitere Informationen über Daten-Handles finden Sie im Abschnitt »Datenverwaltung«.

Das Array sollte eine Struktur für jedes Service-/Themenname-Paar enthalten, das mit dem vom Client angegebenen übereinstimmt. Die DDEML wählt ein Paar zur Einrichtung der Konversation aus und gibt ein Handle an den Client zurück. Dann sendet sie die Transaktion *XTYP_CONNECT_CONFIRM* an den Server, wenn er diese Transaktion nicht filtert.

Das folgende Beispiel zeigt eine typische Antwort eines Server auf die *XTYP_WILDCONNECT*-Transaktion:

```
#define CTOPICS 2

UINT type;
UINT fmt;
HSZPAIR ahp[CTOPICS + 1];
HSZ ahszTopicList[CTOPICS];
HSZ hszServ, hszTopic;
WORD i, j;

if (type == XTYP_WILDCONNECT) {

    /*
     * Themenliste durchsuchen und ein Array mit
     * HSZPAIR-Strukturen erzeugen.
     */

    j = 0;
    for (i = 0; i < CTOPICS; i++) {
        if (hszTopic == (HSZ) NULL ||
                hszTopic == ahszTopicList[i]) {
            ahp[j].hszSvc = hszServ;
            ahp[j++].hszTopic = ahszTopicList[i];
        }
    }

    /*
     * Die Liste wird mit einer HSZPAIR-Struktur beendet, deren
     * Elemente String-Handles mit dem Wert NULL enthalten.
     */

    ahp[j].hszSvc = NULL;
    ahp[j++].hszTopic = NULL;

    /*
     * Das Handle für das globale Speicherobjekt zurückgeben,
     * das die HSZPAIR-Strukturen enthält.
     */

    return DdeCreateDataHandle(
        idInst,          /* Instanzenbezeichner          */
        &ahp,            /* Zeiger auf das HSZPAIR-Array */
        sizeof(HSZ) * j, /* Länge des Array              */
        0,               /* am Anfang beginnen           */
        NULL,            /* NULL-String für Elementnamen */
        fmt,             /* Rückgabe im gleichen Format  */
        0);              /* dem System zuordnen          */
}
```

Sowohl der Client als auch der Server kann eine Konversation jederzeit mit der Funktion *DdeDisconnect* beenden. Dadurch erhält die Callback-Funktion des

Partners in der Konversation eine *XTYP_DISCONNECT*-Transaktion, wenn dieser nicht das Filter-Flag *CBF_SKIP_DISCONNECT* angegeben hat. Normalerweise reagiert eine Anwendung auf die *XTYP_DISCONNECT*-Transaktion, indem sie mit Hilfe der Funktion *DdeQueryConvInfo* Informationen über die beendete Konversation anfordert. Nachdem die Callback-Funktion die *XTYP_DISCONNECT*-Transaktion bearbeitet hat, ist das Handle der Konversation nicht mehr gültig.

Eine Client-Anwendung, die eine *XTYP_DISCONNECT*-Transaktion in ihrer DDE-Callback-Funktion empfängt, kann die Konversation mit der Funktion *DdeReconnect* erneut einrichten. Der Client muß *DdeReconnect* aus seiner DDE-Callback-Funktion heraus aufrufen.

Mehrfach-konversationen

Eine Client-Anwendung kann durch die Funktion *DdeConnectList* ermitteln, ob brauchbare Server im System verfügbar sind. Sie muß beim Aufruf einen Service- und einen Themennamen angeben. Dadurch sendet die DDEML die *XTYP_WILDCONNECT*-Transaktion an die DDE-Callback-Funktionen aller Server, die mit dem Servicenamen übereinstimmen (außer an diejenigen, die die Transaktion filtern). Die Callback-Funktion eines Server sollte ein Daten-Handle zurückliefern, das ein nullterminiertes Array von *HSZPAIR*-Strukturen bezeichnet. Das Array sollte für jedes Service-/Themenname-Paar, das mit dem vom Client angegebenen übereinstimmt, eine Struktur enthalten. Der Server erhält das Handle der Konversation über die *XTYP_CONNECT_CONFIRM*-Transaktion, wenn er diese nicht filtert.

Ein Client kann beim Aufruf der Funktion *DdeConnectList NULL* für den Servicenamen, den Themennamen oder für beide angeben. Wenn der Servicename *NULL* ist, antworten alle Server, die das angegebene Thema unterstützen. Mit jedem von ihnen wird eine Konversation eingerichtet, auch wenn es sich um mehrere Instanzen des gleichen Server handelt. Ist der Themenname *NULL*, wird eine Konversation über jedes Thema eingerichtet, das von den Server-Anwendungen erkannt wird, die mit dem Servicenamen übereinstimmen.

Ein Client kann die Funktionen *DdeQueryNextServer* und *DdeQueryConvInfo* verwenden, um die Server zu identifizieren, die auf die *DdeConnectList*-Funktion antworten. Die Funktion *DdeQueryNextServer* liefert das nächste Konversation-Handle in einer Konversationsliste. Die Funktion *DdeQueryConvInfo* füllt eine *CONVINFO*-Struktur mit Informationen über die Konversation. Der Client kann die benötigten Handles speichern und die restlichen verwerfen.

Das folgende Beispiel verwendet die Funktion *DdeConnectList*, um Konversationen mit allen Server-Anwendungen einzurichten, die das Thema *System* unterstützen. Es verwendet dann die Funktionen *DdeQueryNextServer* und *DdeQueryConvInfo*, um die String-Handles der Servicenamen der Server zu ermitteln und speichert sie in einem Puffer:

```
HCONVLIST hconvlist; /* Konversationsliste */
DWORD idInst;        /* Instanzenbezeichner */
HSZ hszSystem;       /* Thema System */
```

```
HCONV hconv = NULL;  /* Handle der Konversation */
CONVINFO ci;         /* speichert Informationen
                        über die Konversation */
UINT cConv = 0;      /* Zähler der Konversation-Handles */
HSZ *pHsz, *aHsz;    /* Zeiger auf String-Handles */

/* Verbindung mit allen Server-Anwendungen herstellen, die
   das Thema System unterstützen. */

hconvList= DdeConnectList(idInst, NULL, hszSystem, NULL, NULL);

/* Anzahl der Handles in der Konversationsliste zählen. */

while ((hconv = DdeQueryNextServer(hconvList, hconv)) != NULL)
    cConv++;

/* Puffer für die String-Handles reservieren. */

hconv = NULL;
aHsz = (HSZ *) LocalAlloc(LMEM_FIXED, cConv * sizeof(HSZ));

/* Die String-Handles in den Puffer kopieren. */

pHsz = aHsz;
while ((hconv= DdeQueryNextServer(hconvList, hconv)) != NULL) {
    DdeQueryConvInfo(hconv, QID_SYNC, (PCONVINFO) &ci);
    DdeKeepStringHandle(idInst, ci.hszSvcPartner);
    *pHsz++ = ci.hszSvcPartner;
}

.
. /* Die Handles in der Konversation mit dem Server benutzen. */
.

/* Speicher freigeben und Konversation beenden. */

LocalFree((HANDLE) aHsz);
DdeDisconnectList(hconvList);
```

Eine Anwendung kann eine einzelne Konversation in einer Konversationsliste mit der Funktion *DdeDisconnect* und alle Konversationen mit der Funktion *DdeDisconnectList* beenden. Beide Funktionen veranlassen die DDEML, *XTYP_DISCONNECT*-Transaktionen an die DDE-Callback-Funktion jedes Partners zu senden. Die Funktion *DdeDisconnectList* sendet eine *XTYP_DISCONNECT*-Transaktion für jedes Handle in der Liste.

Ein Client kann die Funktion *DdeConnectList* verwenden, um die Konversation-Handles einer Konversationsliste aufzulisten, indem er das Handle für die Konversationsliste an die Funktion übergibt. Der Auflistungsvorgang entfernt die Handles der beendeten Konversationen aus der Liste.

Wenn die Funktion *DdeConnectList* ein existierendes Konversationslisten-Handle angibt und einen Service- oder Themennamen, der verschieden von dem ist, der zur Erstellung der vorhandenen Konversationsliste verwendet wurde, erstellt die Funktion eine neue Konversationsliste, die die Handles aller neuen Konversationen und die Handles aus der bestehenden Liste enthält.

Die Funktion *DdeConnectList* versucht, doppelte Konversationen in einer Konversationsliste zu vermeiden. Eine doppelte Konversation ist eine zweite Konversation mit dem gleichen Server und dem gleichen Service- und Themennamen. Zwei solche Konversationen hätten zwar verschiedene Handles, wären aber Duplikate.

Datenverwaltung

Weil der DDE globalen Hauptspeicher verwendet, um Daten von einer Anwendung an eine andere zu übergeben, stellt die DDEML eine Reihe von Funktionen bereit, mit deren Hilfe DDE-Anwendungen globale Speicherobjekte erzeugen und verwalten können.

Alle Tranksaktionen, die am Datenaustausch beteiligt sind, benötigen einen lokalen Puffer, der die Daten enthält. Dieser Puffer wird von der Anwendung reserviert, die die Daten bereitstellt. Nach der Reservierung muß sie die Funktion *DdeCreateDataHandle* aufrufen. Diese Funktion reserviert ein globales Speicherobjekt, kopiert die Daten vom Puffer in das Speicherobjekt und gibt das Daten-Handle der Anwendung zurück. Ein Daten-Handle ist ein *DWORD*-Wert, durch den die DDEML den Zugriff auf Daten im globalen Speicherobjekt ermöglicht. Um die Daten in einem globalen Speicherobjekt gemeinsam zu benutzen, übergibt eine Anwendung das Daten-Handle an die DDEML. Diese gibt das Handle an die DDE-Callback-Funktion der Anwendung weiter, die die Datentransaktion erhält.

Das folgende Beispiel zeigt, wie ein globales Speicherobjekt erzeugt und ein Handle des Objekts abgerufen wird. Während der *XTYP_ADVREQ*-Transaktion konvertiert die Callback-Funktion die aktuelle Zeit in einen ASCII-String, kopiert den String in einen lokalen Puffer und erzeugt dann ein globales Speicherobjekt, das den String enthält. Die Callback-Funktion gibt das Handle des globalen Speicherobjekts an die DDEML zurück, die das Handle dann der Client-Anwendung übergibt.

```
typedef struct { /* tm */
    int hour;
    int minute;
    int second;
} TIME;

TIME tmTime;
HSZ hszTime;
HSZ hszNow;
HDDEDATA EXPENTRY DdeProc(wType, wFmt, hConv, hsz1, hsz2,
    hData, dwData1, dwData2)
WORD wType;
WORD wFmt;
HCONV hConv;
HSZ hsz1;
```

```
HSZ hsz2;
HDDEDATA hData;
DWORD dwData1;
DWORD dwData2;
{
   char szBuf[32];

   switch (wType) {

     case XTYP_ADVREQ:
      if ((hsz1 == hszTime && hsz2 == hszNow)
            && (wFmt == CF_TEXT)) {

        /* Formatierten Uhrzeit-String in den
           Puffer kopieren. */

        itoa(tmTime.hour, szBuf, 10);
        strcat(szBuf, ":");
        if (tmTime.minute < 10)
           strcat(szBuf, "0");
        itoa(tmTime.minute, &szBuf[strlen(szBuf)], 10);
        strcat(szBuf, ":");
        if (tmTime.second < 10)
            strcat(szBuf, "0");
        itoa(tmTime.second, &szBuf[strlen(szBuf)], 10);
        szBuf[strlen(szBuf)] = '\0';

       /* Globales Speicherobjekt erzeugen und das Daten-Handle
          zurückgeben. */

        return (DdeCreateDataHandle(
        idInst,           /* Instanzenbezeichner               */
        (LPBYTE) szBuf,   /* Quellpuffer                       */
        strlen(szBuf) + 1, /* Größe des globalen Speicherobjekts */
        0L,               /* Offset, ausgehend vom Anfang      */
        hszNow,           /* String mit dem Elementnamen       */
        CF_TEXT,          /* Format der Zwischenablage         */
        0));              /* Keine Flags zur Erzeugung         */
       } else
           return (HDDEDATA) NULL;

      .
      . /* Andere Transaktionstypen bearbeiten. */
      .
   }
}
```

Die empfangende Anwendung erhält einen Zeiger auf das globale Speicherobjekt, indem sie das Daten-Handle an die Funktion *DdeAccessData* übergibt. Der von *DdeAccessData* zurückgelieferte Zeiger ermöglicht einen Lesezugriff. Die Anwendung sollte ihn verwenden, um die Daten einzusehen und dann die Funktion *DdeUnaccessData* aufrufen, um den Zeiger ungültig zu machen. Die

Anwendung kann die Daten mit Hilfe der Funktion *DdeGetData* in einen lokalen Puffer kopieren.

Das folgende Beispiel fordert einen Zeiger auf das globale Speicherobjekt an, das durch den Parameter *hData* bezeichnet wird, kopiert den Inhalt in einen lokalen Puffer und macht dann den Zeiger ungültig.

```
HDDEDATA hData;
LPBYTE lpszAdviseData;
DWORD cbDataLen;
DWORD i;
char szData[32];

case XTYP_ADVDATA:

    lpszAdviseData = DdeAccessData(hData, &cbDataLen);
    for (i = 0; i < cbDataLen; i++)
        szData[i] = *lpszAdviseData++;
    DdeUnaccessData(hData);
    return (HDDEDATA) TRUE;
```

Wenn eine Anwendung ein Daten-Handle erzeugt und an die DDEML übergibt, wird das Handle in der erzeugenden Anwendung normalerweise ungültig. Dies ist hilfreich, wenn die Anwendung Daten mit nur einer Anwendung gemeinsam benutzen will. Wenn eine Anwendung jedoch die Daten mit mehreren Anwendungen gemeinsam benutzen will, sollte sie das *HDATA_APPOWNED*-Flag in *DdeCreateDataHandle* angeben. Dadurch wird sie Besitzer des Speicherobjekts und verhindert, daß die DDEML das Daten-Handle ungültig macht. Wenn die erzeugende Anwendung das Speicherobjekt nicht mehr benötigt, sollte sie es mit der Funktion *DdeFreeDataHandle* freigeben.

Solange eine Anwendung das Handle eines globalen Speicherobjekts noch nicht an die DDEML übergeben hat, kann sie mit der Funktion *DdeAddData* Daten zum Objekt hinzufügen oder Daten im Objekt überschreiben. Normalerweise wird diese Funktion zum Füllen eines nicht initialisierten globalen Speicherobjekts verwendet. Nachdem eine Anwendung ein Daten-Handle an die DDEML übergeben hat, kann das vom Handle bezeichnete Speicherobjekt nicht mehr geändert, sondern nur noch freigegeben werden.

Die DDEML-Datenverwaltungsfunktionen können **huge**-Speicherobjekte handhaben. Eine DDEML-Anwendung sollte die Größe eines globalen Speicherobjekts überprüfen und einen entsprechend großen **huge**-Puffer reservieren, bevor sie das Objekt kopiert.

Transaktionsverwaltung

Nachdem ein Client eine Konversation mit einem Server eingerichtet hat, kann er Transaktionen senden, um Daten und Dienste vom Server anzufordern. Die folgenden Themen in diesem Abschnitt beschreiben die Transaktionstypen, die Clients zur Interaktion mit einem Server verwenden können.

Die Request-Transaktion

Eine Client-Anwendung kann die *XTYP_REQUEST*-Transaktion verwenden, um Daten von einer Server-Anwendung anzufordern. Dazu muß sie die Funktion *DdeClientTransaction* aufrufen, *XTYP_REQUEST* als Transaktionstyp und das benötigte Datenelement angeben.

Die DDEML übergibt die *XTYP_REQUEST*-Transaktion zusammen mit dem vom Client gewünschten Thema, Element und Datenformat an den Server. Wenn der Server das gewünschte Thema, Element und Datenformat unterstützt, sollte er ein Daten-Handle zurückgeben, das den aktuellen Wert des Elements bezeichnet. Die DDEML übergibt dieses Handle als Rückgabewert der Funktion *DdeClientTransaction* an den Client. Der Server sollte *NULL* zurückgeben, wenn er das gewünschte Thema, Element oder Datenformat nicht unterstützt.

Die Funktion *DdeClientTransaction* verwendet den Parameter *lpdwResult*, um ein Status-Flag der Transaktion an den Client zurückzugeben. Wenn der Server die Transaktion *XTYP_REQUEST* nicht bearbeitet, liefert *DdeClientTransaction* *NULL* zurück und *lpdwResult* zeigt auf das Flag *DDE_FNOTPROCESSED* oder *DDE_FBUSY*. Wird das Flag *DDE_FNOTPROCESSED* zurückgegeben, kann der Client nicht bestimmen, warum der Server die Transaktion nicht bearbeitet hat.

Wenn ein Server die Transaktion *XTYP_REQUEST* nicht unterstützt, sollte er das Filter-Flag *CBF_FAIL_REQUESTS* in der Funktion *DdeInitialize* angeben. Dies verhindert, daß die DDEML diese Transaktion an den Server sendet.

Die Poke-Transaktion

Ein Client kann nicht angeforderte Daten an einen Server senden, indem er die Funktion *DdeClientTransaction* verwendet, um eine *XTYP_POKE*-Transaktion an die Callback-Funktion des Server zu übergeben.

Die Client-Anwendung erzeugt einen Puffer, der die an den Server zu sendenden Daten enthält und übergibt dann einen Zeiger auf den Puffer als Argument an die Funktion *DdeClientTransaction*. Als Alternative hierzu kann der Client die Funktion *DdeCreateDataHandle* verwenden, um ein Daten-Handle anzufordern, das die Daten bezeichnet und dieses dann an *DdeClientTransaction* übergeben. In beiden Fällen gibt der Client beim Aufruf der Funktion *DdeClientTransaction* auch den Themennamen, den Elementnamen und das Datenformat an.

Die DDEML übergibt die Transaktion *XTYP_POKE* an den Server und gibt dabei den vom Client gewünschten Themennamen, Elementnamen und das Datenformat an. Wenn er das Datenelement und -format akzeptiert, sollte der Server *DDE_FACK*, wenn er die Daten zurückzuweist, *DDE_FNOTPROCESSED* zurückgeben. Wenn der Server zu beschäftigt ist, um die Daten zu akzeptieren, sollte er *DDE_FBUSY* zurückgeben.

Wenn die Funktion *DdeClientTransaction* zurückkehrt, kann der Client den Parameter *lpdwResult* verwenden, um auf das Status-Flag der Transaktion zuzugreifen. Ist das Flag *DDE_FBUSY*, sollte der Client die Transaktion später noch einmal senden.

Wenn ein Server die Transaktion *XTYP_POKE* nicht unterstützt, sollte er das Filter-Flag *CBF_FAIL_PROCESS* in der Funktion *DdeInitialize* angeben. Dies verhindert, daß die DDEML diese Transaktion an den Server sendet.

Die Advise-Transaktion

Eine Client-Anwendung kann die DDEML verwenden, um eine oder mehrere Verknüpfungen zu Elementen in einer Server-Anwendung herzustellen. Wenn eine solche Verknüpfung hergestellt ist, sendet der Server periodisch Aktualisierungen der verknüpften Elemente an die Client-Anwendung (normalerweise immer dann, wenn sich der Wert des der Server-Anwendung zugeordneten Elements ändert). Dadurch entsteht eine Advise-Schleife zwischen den beiden Anwendungen, die solange besteht, bis sie vom Client beendet wird.

Es gibt zwei Arten von Advise-Schleifen: *heiß* und *warm*. In einer heißen Advise-Schleife sendet der Server sofort ein Daten-Handle, das den geänderten Wert bezeichnet. In einer warmen Schleife benachrichtigt der Server den Client, daß sich der Wert des Datenelements geändert hat, sendet aber das Daten-Handle erst dann, wenn es vom Client angefordert wird. Ein Client kann eine heiße Advise-Schleife mit einem Server anfordern, indem er den Transaktionstyp *XTYP_ADVSTART* in einem Aufruf der Funktion *DdeClientTransaction* angibt. Um eine warme Advise-Schleife anzufordern, muß der Client das Flag *XTYPF_NODATA* mit dem Transaktionstyp *XTYP_ADVSTART* kombinieren. In beiden Fällen übergibt die DDEML die *XTYP_ADVSTART*-Transaktion an die DDE-Callback-Funktion des Server. Diese Funktion sollte die Parameter untersuchen, die die *XTYP_ADVSTART*-Transaktion begleiten (einschließlich des gewünschten Formats, Themennamens und Elementnamens) und dann *TRUE* zurückgeben, um die Schleife zuzulassen oder *FALSE*, um sie abzulehnen.

Nachdem eine Advise-Schleife eingerichtet wurde, sollte die Server-Anwendung die Funktion *DdePostAdvise* immer dann aufrufen, wenn sich der Wert des Elements ändert, das dem angeforderten Elementnamen zugeordnet ist. Daraus ergibt sich die Übergabe einer *XTYP_ADVREQ*-Transaktion an die eigene DDE-Callback-Funktion des Server. Die DDE-Callback-Funktion des Server sollte ein Daten-Handle zurückliefern, das den neuen Wert des Elements bezeichnet. Die DDEML benachrichtigt dann den Client über die Änderung des angegebenen Elements, indem sie die *XTYP_ADVDATA*-Transaktion an die DDE-Callback-Funktion des Client sendet.

Wenn der Client eine heiße Advise-Schleife angefordert hat, übergibt die DDEML das Daten-Handle für das geänderte Element mit der *XTYP_ADVDATA*-Transaktion an den Client. Andernfalls kann der Client eine *XTYP_REQUEST*-Transaktion senden, um das Daten-Handle anzufordern.

Ein Server kann Aktualisierungen möglicherweise schneller senden, als ein Client die neuen Daten bearbeiten kann. Dies kann für einen Client problematisch werden, der umfangreiche Bearbeitungen der Daten vornehmen muß. In diesem Fall sollte der Client das *XTYPF_ACKREQ*-Flag angeben, wenn er eine Advise-Schleife anfordert. Dadurch wartet der Server darauf, daß der Client den Empfang und die Bearbeitung eines Datenelements bestätigt, bevor er das

nächste Element sendet. Advise-Schleifen, die mit dem Flag *XTYPF_ACKREQ* eingerichtet werden, sind bei schnellen Servern robuster, können aber gelegentlich Aktualisierungen übergehen. Ohne dieses Flag eingerichtete Advise-Schleifen übergehen garantiert keine Aktualisierungen, solange die Verbindung zum Server besteht.

Ein Client kann eine Advise-Schleife durch Angabe des Transaktionstyps *XTYP_ADVSTOP* in einem Aufruf der Funktion *DdeClientTransaction* beenden. Wenn ein Server keine Advise-Schleifen unterstützt, sollte er das Filter-Flag *CBF_FAIL_ADVISES* in der Funktion *DdeInitialize* angeben. Dies verhindert, daß die DDEML die Transaktionen *XTYP_ADVSTART* und *XTYP_ADVSTOP* an den Server sendet.

Die Execute-Transaktion

Ein Client kann die *XTYP_EXECUTE*-Transaktion verwenden, um einen Server zu veranlassen, einen oder mehrere Befehle auszuführen. Dazu erzeugt der Client zuerst einen Puffer, der den Befehls-String für den Server enthält und übergibt dann beim Aufruf der Funktion *DdeClientTransaction* einen Zeiger auf den Puffer oder ein Daten-Handle, das den Puffer bezeichnet. Andere benötigte Parameter sind das Handle der Konversation, das String-Handle des Elementnamens, die Formatangabe und der Transaktionstyp *XTYP_EXECUTE*. Erzeugt eine Anwendung ein Daten-Handle zur Übergabe von Befehlsdaten, muß sie für den Parameter *hszItem* der Funktion *DdeCreateDataHandle NULL* angeben.

Die DDEML übergibt die Transaktion *XTYP_EXECUTE* an die DDE-Callback-Funktion des Server und gibt dabei den Formatnamen, das Konversation-Handle, den Themennamen und das Daten-Handle an, das den Befehls-String bezeichnet. Wenn der Server den Befehl unterstützt, sollte er die Funktion *DdeAccessData* verwenden, um einen Zeiger auf den Befehls-String anzufordern, den Befehl ausführen und dann *DDE_FACK* zurückliefern. Wenn der Server den Befehl nicht unterstützt oder die Transaktion nicht vollenden kann, sollte er *DDE_FNOTPROCESSED* zurückgeben. Ist er zu beschäftigt, um die Transaktion zu vollenden, sollte er *DDE_FBUSY* zurückgeben.

Wenn die Funktion *DdeClientTransaction* zurückkehrt, kann der Client den Parameter *lpdwResult* verwenden, um auf das Status-Flag der Transaktion zuzugreifen. Ist das Flag *DDE_FBUSY*, sollte der Client die Transaktion später noch einmal senden.

Wenn ein Server die Transaktion *XTYP_EXECUTE* nicht unterstützt, sollte er das Filter-Flag *CBF_FAIL_EXECUTES* in der Funktion *DdeInitialize* angeben. Dies verhindert, daß die DDEML diese Transaktion an den Server sendet.

Synchrone und asynchrone Transaktionen

Eine Client kann entweder synchrone oder asynchrone Transaktionen senden. In einer synchronen Transaktion legt der Client einen Timeout-Wert fest,

der die maximale Zeitspanne angibt, die auf die Bearbeitung der Transaktion durch den Server gewartet wird. Die Funktion *DdeClientTransaction* kehrt solange nicht zurück, bis der Server die Transaktion bearbeitet hat, die Transaktion fehlgeschlagen ist oder der Timeout-Wert erreicht ist. Der Client gibt den Timeout-Wert beim Aufruf der Funktion *DdeClientTransaction* an.

Während einer synchronen Transaktion wartet der Client in einer modalen Schleife darauf, daß die Transaktion bearbeitet wird. Der Client kann immer noch Benutzereingaben bearbeiten. Er kann aber keine weiteren synchronen Transaktion senden, bis die Funktion *DdeClientTransaction* zurückkehrt.

Ein Client sendet eine asynchrone Transaktion, indem er das Flag *TIMEOUT_ASYNC* in der Funktion *DdeClientTransaction* angibt. Die Funktion kehrt zurück, nachdem die Transaktion begonnen hat und übergibt dabei einen Transaktionsbezeichner an den Client. Wenn der Server die Bearbeitung der asynchronen Transaktion beendet, sendet die DDEML eine *XTYP_XACT_COMPLETE*-Transaktion an den Client. Einer der Parameter, den die DDEML während der *XTYP_XACT_COMPLETE*-Transaktion an den Client übergibt, ist der Transaktionsbezeichner. Indem er diesen mit dem von der Funktion *DdeClientTransaction* zurückgegebenen Bezeichner vergleicht, ermittelt der Client, welche asynchrone Transaktion vom Server beendet wurde.

Ein Client kann die Funktion *DdeSetUserHandle* als Hilfe zum Bearbeiten asynchroner Transaktionen verwenden. Mit Hilfe dieser Funktion kann ein Client einem Konversation-Handle und einem Transaktionsbezeichner einen anwendungsdefinierten *DWORD*-Wert zuordnen. Der Client kann den anwendungsdefinierten *DWORD*-Wert mit Hilfe der Funktion *DdeQueryConvInfo* während einer *XTYP_XACT_COMPLETE*-Transaktion ermitteln. Dadurch muß eine Anwendung keine Liste der aktiven Transaktionsbezeichner verwalten.

Wenn ein Server eine asynchrone Transaktion nicht rechtzeitig bearbeitet, kann der Client die Transaktion durch die Funktion *DdeAbandonTransaction* abbrechen. Die DDEML gibt alle der Transaktion zugeordneten Ressourcen frei und verwirft die Ergebnisse der Transaktion, wenn der Server die Bearbeitung beendet.

Die asynchrone Transaktion ist für Anwendungen bestimmt, die viele DDE-Transaktionen senden müssen, während sie gleichzeitig viele Bearbeitungen, wie z.B. Berechnungen, durchführen. Die asynchrone Methode ist auch in Anwendungen nützlich, die die Bearbeitung von DDE-Transaktionen zeitweise unterbrechen müssen, um andere Aufgaben störungsfrei beenden zu können. In den meisten anderen Situationen sollte eine Anwendung die synchrone Methode verwenden.

Synchrone Transaktionen sind einfacher zu verwalten und schneller als asynchrone Transaktionen. Es kann jedoch nur eine synchrone Transaktion zur Zeit ausgeführt werden, während viele asynchrone Transaktionen zur gleichen Zeit möglich sind. Durch synchrone Transaktionen kann ein langsamer Server einen Client veranlassen, untätig zu bleiben, während er auf eine Antwort wartet. Synchrone Transaktionen bewirken außerdem, daß der Client eine modale Schleife betritt, die das Filtern von Botschaften in der eigenen Botschaftsschleife der Anwendung umgehen kann.

Transaktions-steuerung

Eine Anwendung kann Transaktionen an ihre DDE-Callback-Funktion vorübergehend ausschließen – entweder die einem bestimmten Konversation-Handle zugeordneten Transaktionen oder alle Transaktionen, ohne Berücksichtigung dieses Handle. Dies ist nützlich, wenn eine Anwendung eine Transaktion erhält, die eine längere Bearbeitung erfordert. In diesem Fall kann eine Anwendung *CBR_BLOCK* zurückgeben, um spätere Transaktionen auszuschließen, die dem Konversation-Handle dieser Transaktion zugeordnet sind. Die Bearbeitung anderer Konversationen bleibt möglich.

Wenn die Bearbeitung vollendet ist, ruft die Anwendung die Funktion *DdeEnableCallback* auf, um Transaktionen wieder zuzulassen, die der gesperrten Konversation zugeordnet sind. Der Aufruf von *DdeEnableCallback* veranlaßt die DDEML zum erneuten Senden der Transaktion, die zum Sperren der Konversation geführt hat. Deshalb sollte die Anwendung das Ergebnis der Transaktion so speichern, daß sie es ohne erneute Bearbeitung der Transaktion ermitteln und zurückgeben kann.

Eine Anwendung kann alle einem bestimmten Konversation-Handle zugeordneten Transaktionen sperren, indem sie das Handle und das Flag *EC_DISABLE* in einem Aufruf der Funktion *DdeEnableCallback* angibt. Durch Angabe eines *NULL*-Handle kann eine Anwendung alle Transaktionen für alle Konversationen sperren.

Wenn eine Konversation gesperrt wird, speichert die DDEML Transaktionen für die Konversation in einer Transaktionswarteschlange. Sobald die Anwendung die Konversation reaktiviert, entnimmt die DDEML die gespeicherten Transaktionen der Warteschlange und übergibt sie an die entsprechende Callback-Funktion. Obwohl die Kapazität der Transaktionswarteschlange groß ist, sollte eine Anwendung eine gesperrte Konversation sobald wie möglich reaktivieren, um den Verlust von Transaktionen zu verhindern.

Eine Anwendung kann die normale Transaktionsbearbeitung wieder aufnehmen, indem sie das Flag *EC_ENABLEALL* in der Funktion *DdeEnableCallback* angibt. Eine kontrolliertere Wiederaufnahme der Transaktionsbearbeitung kann die Anwendung mit dem Flag *EC_ENABLEONE* erreichen. Es entfernt eine Transaktion aus der Transaktionswarteschlange und übergibt sie an die entsprechende Callback-Funktion. Nachdem die einzelne Transaktion bearbeitet ist, werden alle Konversationen erneut deaktiviert.

Transaktions-klassen

Die DDEML besitzt vier Transaktionsklassen. Jede Klasse wird durch eine Konstante bezeichnet, die mit dem Präfix *XCLASS_* beginnt. Die Klassen sind in der DDEML-Header-Datei definiert. Die Klassenkonstante wird mit der Transaktionstypkonstante kombiniert und an die DDE-Callback-Funktion der empfangenden Anwendung übergeben.

Die Klasse einer Transaktion bestimmt den Rückgabewert, der von einer

Callback-Funktion erwartet wird, wenn sie die Transaktion bearbeitet. Die folgende Tabelle zeigt die Rückgabewerte und Transaktionstypen, die jeder der vier Transaktionsklassen zugeordnet sind:

Klasse	Rückgabewert	Transaktion
XCLASS_BOOL	*TRUE* oder *FALSE*	*XTYP_ADVSTART* *XTYP_CONNECT*
XCLASS_DATA	Ein Daten-Handle, *CBR_BLOCK* oder *NULL*	*XTYP_ADVREQ* *XTYP_REQUEST* *XTYP_WILDCONNECT*
XCLASS_FLAGS	Ein Transaktions-Flag: *DDE_FACK*, *DDE_FBUSY* oder *DDE_FNOTPROCESSED*	*XTYP_ADVDATA* *XTYP_EXECUTE* *XTYP_POKE*
XCLASS_NOTIFICATION	Keiner	*XTYP_ADVSTOP* *XTYP_CONNECT_CONFIRM* *XTYP_DISCONNECT* *XTYP_ERROR* *XTYP_REGISTER* *XTYP_UNREGISTER* *XTYP_XACT_COMPLETE*

Übersicht

Die folgende Liste enthält alle DDE-Transaktionstypen, den Empfänger jedes Typs und eine Beschreibung der Aktivitäten, die die DDEML veranlassen, die Typen zu erzeugen:

Transaktionstyp	Empfänger	Grund
XTYP_ADVDATA	*Client*	Ein Server hat auf eine *XTYP_ADVREQ*-Transaktion durch Rückgabe eines Daten-Handle geantwortet.
XTYP_ADVREQ	*Server*	Ein Server hat die Funktion *DdePostAdvise* aufgerufen und dadurch angezeigt, daß sich der Wert eines Datenelements in einer Advise-Schleife geändert hat.
XTYP_ADVSTART	*Server*	Ein Client hat den Transaktionstyp *XTYP_ADVSTART* in einem Aufruf der Funktion *DdeClientTransaction* angegeben.
XTYP_ADVSTOP	*Server*	Ein Client hat den Transaktionstyp *XTYP_ADVSTOP* in einem Aufruf der Funktion *DdeClientTransaction* angegeben.

Transaktionstyp	Empfänger	Grund
XTYP_CONNECT	*Server*	Ein Client hat die Funktion *DdeConnect* aufgerufen und dabei einen vom Server unterstützten Service- und Themennamen angegeben.
XTYP_CONNECT_-CONFIRM	*Server*	Der Server hat *TRUE* als Antwort auf eine *XTYP_CONNECT*- oder *XTYP_WILDCONNECT*-Transaktion zurückgeliefert.
XTYP_DISCONNECT	*Client/Server*	Ein Partner in einer Konversation hat die Funktion *DdeDisconnect* aufgerufen und bewirkt dadurch, daß beide Partner die Transaktion erhalten.
XTYP_ERROR	*Client/Server*	Ein kritischer Fehler ist aufgetreten. Die DDEML hat möglicherweise nicht genug Ressourcen, um fortzufahren.
XTYP_EXECUTE	*Server*	Ein Client hat den Transaktionstyp *XTYP_EXECUTE* in einem Aufruf der Funktion *DdeClientTransaction* angegeben.
XTYP_MONITOR	*DDE-überwachende Anwendung*	Ein DDE-Ereignis ist im System aufgetreten. Mehr Informationen über DDE-überwachende Anwendungen können Sie dem Abschnitt »Überwachende Anwendungen« entnehmen.
XTYP_POKE	*Server*	Ein Client hat den Transaktionstyp *XTYP_-POKE* in einem Aufruf der Funktion *DdeClientTransaction* angegeben.
XTYP_REGISTER	*Client/Server*	Eine Server-Anwendung hat die Funktion *DdeNameService* zur Registrierung eines Servicenamens verwendet.
XTYP_REQUEST	*Server*	Ein Client hat den Transaktionstyp *XTYP_-REQUEST* in einem Aufruf der Funktion *DdeClientTransaction* angegeben.
XTYP_UNREGISTER	*Client/Server*	Eine Server-Anwendung hat die Funktion *DdeNameService* verwendet, um die Registrierung eines Servicenamens aufzuheben.
XTYP_WILD-CONNECT	*Server*	Ein Client hat die Funktion *DdeConnect* oder *DdeConnectList* aufgerufen und dabei *NULL* für den Service-, den Themennamen oder für beide angegeben.
XTYP_XACT_-COMPLETE	*Client*	Eine asynchrone Transaktion ist beendet worden, die gesendet wurde, als das *TIMEOUT_ASYNC*-Flag in einem Aufruf der Funktion *DdeClientTransaction* angegeben wurde.

Fehler-erkennung

Wenn eine DDEML-Funktion fehlschlägt, kann eine Anwendung die Funktion *DdeGetLastError* aufrufen, um den Grund festzustellen. Die Funktion *DdeGetLastError* liefert einen Fehlerwert zurück, der den Grund des letzten Fehlers bestimmt.

Überwachende Anwendungen

Das Windows-Programm DDESpy (DDESPY.EXE) überwacht DDE-Aktivitäten im System. Sie können DDESpy als Hilfsprogramm beim Testen Ihrer DDE-Anwendungen verwenden.

Sie können die API-Elemente der DDEML verwenden, um Ihre eigenen DDE-überwachenden Anwendungen zu erzeugen. Wie jede DDEML-Anwendung enthält eine DDE-überwachende Anwendung eine DDE-Callback-Funktion. Die DDEML benachrichtigt diese Funktion immer dann, wenn ein DDE-Ereignis auftritt und übergibt dabei Informationen über das Ereignis. Die Anwendung zeigt die Informationen normalerweise in einem Fenster an oder schreibt sie in eine Datei.

Um Benachrichtigungen von der DDEML zu erhalten, muß sich eine Anwendung als DDE-Überwachungsprogramm registrieren, indem sie das Flag *APPCLASS_MONITOR* in einem Aufruf der Funktion *DdeInitialize* angibt. Im gleichen Aufruf kann die Anwendung ein oder mehrere Überwachungs-Flags angeben, um die Ereignistypen zu bestimmen, von denen die DDEML die Callback-Funktion der Anwendung benachrichtigen soll. Die folgende Tabelle beschreibt jedes Überwachungs-Flag, das eine Anwendung angeben kann:

Flag	Bedeutung
MF_CALLBACKS	Benachrichtigt die Callback-Funktion, wenn eine Transaktion an eine DDE-Callback-Funktion im System gesendet wird.
MF_CONV	Benachrichtigt die Callback-Funktion, wenn eine Konversation eingerichtet oder beendet wird.
MF_ERRORS	Benachrichtigt die Callback-Funktion, wenn ein DDEML-Fehler auftritt.
MF_HSZ_INFO	Benachrichtigt die Callback-Funktion, wenn eine DDEML-Anwendung den Verwendungszähler eines String-Handle erzeugt, freigibt oder erhöht, oder wenn ein String-Handle als Ergebnis eines Aufrufs der Funktion *DdeUninitialize* freigegeben wird.
MF_LINKS	Benachrichtigt die Callback-Funktion, wenn eine Advise-Schleife gestartet oder beendet wird.

Flag	Bedeutung
MF_POSTMSGS	Benachrichtigt die Callback-Funktion, wenn das System oder eine Anwendung eine DDE-Botschaft übergibt.
MF_SENDMSGS	Benachrichtigt die Callback-Funktion, wenn das System oder eine Anwendung eine DDE-Botschaft sendet.

Das folgende Beispiel zeigt, wie eine DDE-überwachende Anwendung registriert wird, so daß ihre DDE-Callback-Funktion Benachrichtigungen über alle DDE-Ereignisse erhält:

```
DWORD idInst;
PFNCALLBACK lpDdeProc;
hInst = hInstance;

lpDdeProc = (PFNCALLBACK) MakeProcInstance(
    (FARPROC) DDECallback,  /* zeigt auf die Callback-Funktion */
    hInstance);             /* Instanzen-Handle */

if (DdeInitialize(
      (LPDWORD) &idInst,    /* Instanzenbezeichner */
      lpDdePorc,            /* zeigt auf die Callback-Funktion */
      APPCLASS_MONITOR |    /* es handelt sich um eine
                               überwachende Anwendung */
      MF_CALLBACKS     |    /* Callback-Funktion überwachen */
      MF_CONV          |    /* Konversationsdaten überwachen */
      MF_ERRORS        |    /* DDEML-Fehler überwachen */
      MF_HSZ_INFO      |    /* Daten-Handle-Aktivitäten
                               überwachen */
      MF_LINKS         |    /* Advise-Schleifen überwachen */
      MF_POSTMSGS      |    /* übergebene DDE-Botschaften
                               überwachen */
      MF_SENDMSGS,          /* gesendete DDE-Botschaften
                               überwachen */
      0L))                  /* reserviert */

    return FALSE;
```

Die DDEML informiert eine überwachende Anwendung von einem DDE-Ereignis, indem sie eine *XTYP_MONITOR*-Transaktion an die DDE-Callback-Funktion der Anwendung sendet. Während dieser Transaktion übergibt die DDEML ein Überwachungs-Flag, das den Typ des aufgetretenen DDE-Ereignisses angibt und ein Handle eines globalen Speicherobjekts, das detaillierte Informationen über das Ereignis enthält.

Die DDEML stellt Strukturen bereit, die die Anwendung verwenden kann, um die Informationen aus dem Speicherobjekt zu entnehmen. Für jedes DDE-Ereignis gibt es eine entsprechende Struktur. Die folgende Tabelle beschreibt diese Strukturen.

Struktur	Beschreibung
MONCBSTRUCT	Enthält Informationen über eine Transaktion.
MONCONVSTRUCT	Enthält Informationen über eine Konversation.
MONERRSTRUCT	Enthält Informationen über den letzten DDE-Fehler.
MONLINKSTRUCT	Enthält Informationen über eine Advise-Schleife.
MONHSZSTRUCT	Enthält Informationen über ein String-Handle.
MONMSGSTRUCT	Enthält Informationen über eine DDE-Botschaft, die übergeben oder gesendet wurde.

Das folgende Beispiel zeigt die DDE-Callback-Funktion einer DDE-überwachenden Anwendung, die Informationen über jedes String-Handle-Ereignis formatiert und die Informationen in einem Fenster anzeigt. Die Funktion verwendet die *MONHSZSTRUCT*-Struktur, um die Informationen aus dem globalen Speicherobjekt zu entnehmen.

```
HDDEDATA CALLBACK DDECallback(wType, wFmt, hConv, hsz1, hsz2,
    hData, dwData1, dwData2)
WORD wtype;
WORD wFmt;
HCONV hConv;
HSZ hsz1;
HSZ hsz2;
HDDEDATA hData;
DWORD dwData1;
DWORD dwData2;
{
    LPVOID lpData;
    char *szAction;
    char buf[256];
    DWORD cb;

    switch (wType) {
        case XTYP_MONITOR:

          /* Einen Zeiger auf das globale Speicherobjekt ermitteln */

          if (lpData = DdeAccessData(hData, &cb)) {

          /* Das überwachungs-Flag untersuchen */

    switch (dwData2) {
        case MF_HSZ_INFO:

#define PHSZS ((MONHSZSTRUCT FAR *) lpData)

         / * Das globale Speicherobjekt enthält String-Handle-
           * Daten. Benutzen Sie die MONHSZSTRUCT-Struktur, um
           * auf diese Daten zuzugreifen. */
```

```
                switch (PHSZS->fsAction) {

                    /* Untersuchen Sie das Aktions-Flag,
                     * um die durchzuführende Aktion zu
                     * bestimmen.
                     */

                    case MH_CREATE:
                        szAction = "Created";
                        break;

                    case MH_KEEP:
                        szAction = "Incremented";
                        break;

                    case MH_DELETE:
                        szAction = "Deleted";
                        break;

                    case MH_CLEANUP:
                        szAction = "Cleaned up";
                        break;

                    default:
                        DdeUnaccessData(hData);
                        return ((HDDEDATA) 0);
                }

                /* Formatierte Ausgaben in einen Puffer
                   schreiben. */

                wsprintf(buf,
                   "Handle %    s, Task: % x, HSZ: % lx(% s)",
                   (LPSTR) szAction, PHSZS->Task,
                   PHSZS->hsz, (LPSTR) PHSZS->str);
               .
               . /* Text in einem Fenster anzeigen oder in eine
               .    Datei schreiben. */


               break;
#undef PHSZS

               .
               . /* Weitere MF_*-Flags bearbeiten. */
               .
               default:
                   break;
           }
       }
       /* Das globale Speicherobjekt freigeben. */
```

```
            DdeUnaccessData(hData);
            break;

         default:
            break;
      }
      return ((HDDEDATA) 0);
   }
```

3. OLE-Bibliotheken

Dieses Kapitel beschreibt die Implementierung der Objektverknüpfung und -einbettung (OLE) für Anwendungen, die unter Microsoft Windows laufen. Das Kapitel beschreibt außerdem, wie eine Anwendung verknüpfte und eingebettete Objekte verwendet, um Verbunddokumente zu erzeugen. Folgende Themen bieten zusätzliche Informationen:

- Dynamischer Datenaustausch (DDE)
- Zwischenablage
- Registrierungsdatenbank
- Dynamische Linkbibliotheken
- MDI (Multiple Document Interface)

In diesem Kapitel wird nicht näher auf die empfohlene Benutzerschnittstelle für Anwendungen eingegangen, die verknüpfte und eingebettete Objekte verwenden.

Grundlagen der Objektverknüpfung und -einbettung

Dieser Abschnitt erklärt die grundlegenden Konzepte der OLE und vergleicht deren Möglichkeiten mit denen der Bibliothek für die Verwaltung des dynamischen Datenaustauschs (DDEML).

Verbunddokumente

Eine OLE-Anwendung kann zusammen mit anderen OLE-Anwendungen ein Dokument erstellen, das unterschiedliche Arten von Daten enthält. Diese Daten können vom Benutzer in einfacher Weise bearbeitet werden. Der Benutzer kann das Dokument erweitern, indem er die besten Merkmale der verschiedenen Anwendungen auswählt. OLE-Anwendungen ermöglichen eine Arbeit, die stärker als bisher auf das Dokument und weniger auf die Anwendung selbst bezogen

ist. Beim anwendungsbezogenen Arbeiten steht meist eine einzelne Anwendung im Mittelpunkt. Wird dagegen dokumentbezogen gearbeitet, kann der Benutzer die Vorteile vieler Werkzeuge nutzen, um eine Aufgabe zu lösen.

Ein OLE-Dokument kann viele verschiedene Arten von Daten in unterschiedlichen Formaten enthalten. Ein solches Dokument wird auch Verbunddokument genannt. Ein Verbunddokument nutzt die Vorteile der verschiedenen OLE-Anwendungen, um die unterschiedlichen Daten zu bearbeiten, die am Bildschirm angezeigt werden. Jede Art von Datenformaten kann in einem Verbunddokument zusammengefaßt werden. Eine OLE-Anwendung kann sogar Datenformate unterstützen, die erst neu erstellt worden sind und benötigt dafür wenig oder keinen zusätzlichen Programmcode. Der Benutzer muß nicht wissen, welche Datenformate untereinander kompatibel sind oder wie man eine Anwendung findet und startet, die die Daten erzeugt hat. Wenn ein Benutzer mit einem Teil des Verbunddokuments arbeiten möchte, startet die für diesen Teil des Verbunddokuments verantwortliche Anwendung automatisch.

Eine Broschüre wäre ein Beispiel für ein Verbunddokument, das Text, Tabellen, Zellenbereiche eines Arbeitsblatts und Illustrationen enthält. Diese Informationen könnten im Dokument eingebettet sein oder das Dokument selbst Verbindungen zu den entsprechenden Informationen besitzen, ohne daß letztere im Dokument enthalten sind. Der Benutzer, der mit einer solchen Broschüre arbeitet, könnte automatisch zwischen den Anwendungen umschalten, die die Komponenten des Dokuments erzeugt haben.

Die folgenden Erläuterungen erklären die Beziehungen zwischen einem Verbunddokument und seinen verknüpften und eingebetteten Objekten (siehe Abbildung 3.1)

Verknüpfte und eingebettete Objekte

Ein Objekt besteht aus beliebigen Daten, die sich in einem Verbunddokument befinden und vom Benutzer bearbeitet werden können. Ein Objekt kann eine einzelne Zelle eines Arbeitsblatts oder auch ein komplettes Dokument sein. Wenn ein Objekt in ein Dokument eingefügt wird, verwaltet es die Beziehung zu der Anwendung, die es erzeugt hat. Diese Beziehung kann eine Verknüpfung oder die Einbettung des Objekts in die Datei sein.

Falls das Objekt verknüpft wird, stellt das Dokument nur einen minimalen Speicherbereich für die Daten zur Verfügung, mit denen das Objekt verknüpft ist. Das Objekt kann jedesmal automatisch aktualisiert werden, wenn sich die Daten in der ursprünglichen Anwendung ändern. Soll zum Beispiel ein Bereich der Zellen eines Arbeitsblatts mit den Informationen in einer Textdatei verknüpft werden, würden die entsprechenden Daten in einer anderen Datei abgelegt. Nur die Verknüpfung auf die Daten würde zusammen mit der Textdatei gespeichert.

Falls ein Objekt eingebettet wird, werden alle zugehörigen Daten als Teil der Datei gespeichert, in die das Objekt eingebettet ist. Soll ein Bereich von Zellen eines Arbeitsblatts in eine Textdatei eingebettet werden, würden die Daten der Zellen zusammen mit den notwendigen Formeln mit der Textdatei abgelegt

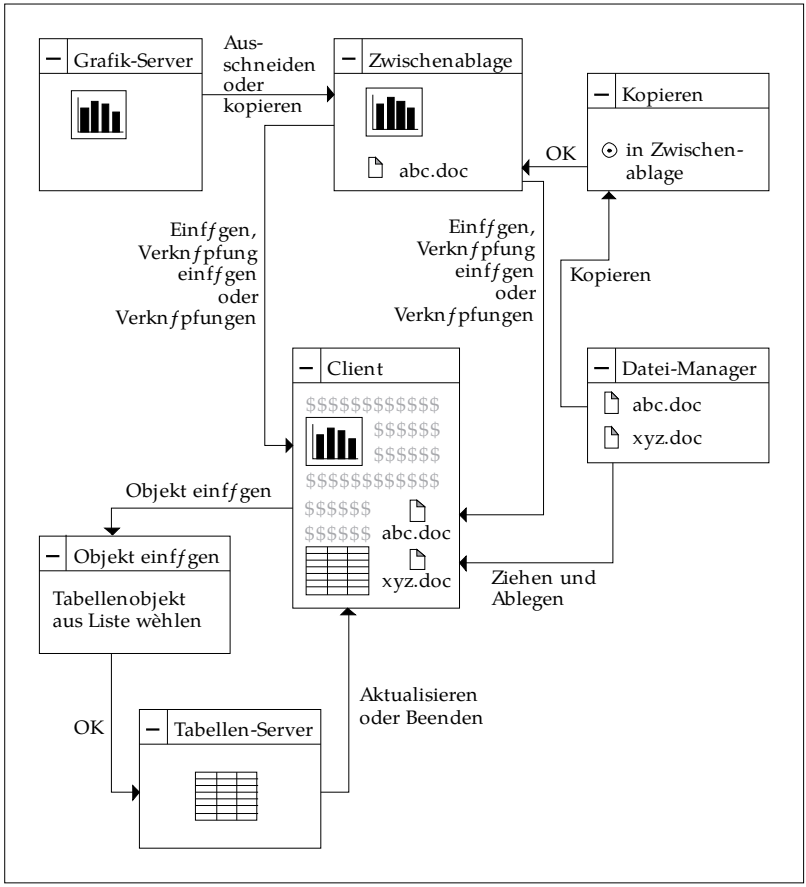


Abbildung 3.1

werden. Der Name der Server-Anwendung wird ebenfalls mit den Daten abgespeichert. Der Benutzer kann dieses Objekt auswählen, während er mit der Textdatei arbeitet. Die zum Arbeitsblatt gehörende Anwendung wird dann automatisch zum Bearbeiten der Zellen des Arbeitsblatts gestartet. Die Daten eines verknüpften oder eines eingebetteten Objekts werden in gleicher Weise dargestellt und verhalten sich auch gleich.

Pakete

Ein Paket ist ein besonderes OLE-Objekt, das ein anderes Objekt, eine Datei oder eine Kommandozeile in eine grafische Darstellung (beispielsweise ein Symbol oder ein Bitmap) einkapselt. Sobald der Benutzer das grafische Objekt doppelt anklickt, aktivieren die OLE-Bibliotheken das Objekt im Paket. Das Paket selbst ist immer ein eingebettetes und niemals ein verknüpftes Objekt. Der Inhalt eines Paketes besteht aus einem eingebetteten Objekt, aus einer Verknüpfung oder aus einer Datei, die aus dem Datei-Manager von Windows abgelegt wurde.

Pakete sind nützlich, um einen umfassenden Überblick über große Dateien oder OLE-Objekte bereitzustellen. Eine Anwendung kann ein Paket wie eine hypertextartige Verbindung benutzen, um Informationen in unterschiedlichen Dokumenten zu verbinden.

Die Version 3.1 von Windows enthält die Anwendung Objekt-Manager (PACKAGER.EXE) von Microsoft Windows. Mit dem Objekt-Manager kann ein Benutzer eine Datei oder ausgewählte Daten mit einem Symbol oder einer Grafik verbinden.

Aktionsarten

Die verschiedenen Bearbeitungstypen, die ein Benutzer an einem Objekt ausführen kann, werden Aktionsarten (englisch: *verb*) genannt. Zwei typische Aktionsarten für ein Objekt sind Play (Abspielen) und Edit (Bearbeiten).

Die Objektart bestimmt das Verhalten des Objekts bei seiner Bearbeitung. Die häufigste Verwendung eines Objekts, z.B. bei einer Sprachsequenz oder einer Animationssequenz, besteht im Abspielen. Ein Benutzer könnte zum Beispiel eine Animationssequenz starten, indem er sie doppelt anklickt. In diesem Fall wäre Play die primäre Aktionsart des Objekts. Eine weitere Möglichkeit ist das Bearbeiten von Objekten. Im Fall eines Texts, der durch ein Textverarbeitungsprogramm erzeugt wurde, heißt die primäre Aktionsart meistens Edit.

Die Client-Anwendung bestimmt normalerweise die primäre Aktionsart, sobald der Benutzer ein Objekt doppelt anklickt. Trotzdem legt die Server-Anwendung die Bedeutung der Aktion fest. Ein Benutzer kann die nachgeordneten Aktionsarten eines Objekts aktivieren, indem er den Befehl *<Klassenname>Objekt* oder das Dialogfenster *Verknüpfungen* verwendet. Nähere Informationen zu diesem Thema enthält der Abschnitt »Die Benutzerschnittstelle der Client-Anwendung«.

Wenn ein Benutzer ein Paket doppelt anklickt, wird die primäre Aktion des Objekts innerhalb dieses Pakets ausgeführt. Die sekundäre Aktionsart eines gepackten Objekts heißt Package edit (Paket bearbeiten). Sobald der Benutzer diese Aktionsart auswählt, wird der Objekt-Manager gestartet. Der Benutzer kann den Objekt-Manager verwenden, um Zugriff auf die sekundäre Aktionsart des Objekts innerhalb des Pakets zu erhalten.

Viele Objekte unterstützen nur eine Aktionsart. So könnte etwa ein Objekt, das durch einen Texteditor erzeugt wurde, nur die Aktionsart Edit unterstützen. Falls ein Objekt nur eine Aktionsart unterstützt, wird diese unabhängig davon verwendet, welche Aktionsart die Client-Anwendung angibt. Nähere Informationen über Aktionsarten enthält der Abschnitt »Registrierung«.

Vorteile der OLE

Die OLE bietet folgende Vorteile.

- Eine Anwendung kann darauf spezialisiert werden, eine bestimmte Arbeit auszuführen. So benötigt eine Zeichenanwendung, die OLE implementiert, keine Werkzeuge zum Bearbeiten von Text. Ein Benutzer könnte Text in die Zeichnung eintragen und mit Hilfe eines Texteditors bearbeiten, der die OLE unterstützt.

- Eine Anwendung wird automatisch für zukünftige Datenformate erweiterbar, da der Inhalt eines Objekts nicht von dem Dokument abhängt, das das Objekt enthält.
- Der Benutzer kann sich ausschließlich auf die Aufgabe konzentrieren und muß sich nicht um die Software kümmern, die dafür benötigt wird.
- Die Dateien sind viel kleiner, da die Verknüpfung mit Objekten einer Datei die Möglichkeit gibt, das Objekt zu benutzen, ohne selbst die Daten des Objekts speichern zu müssen.
- Ein Dokument kann gedruckt oder übermittelt werden, ohne die Anwendung aufzurufen, die das Dokument ursprünglich erzeugt hat.
- Verknüpfte Objekte können dynamisch aktualisiert werden.

Zukünftige Implementierungen dieses Protokolls werden mit einer noch größeren Anzahl von Objekttypen arbeiten können. Der Benutzer einer Anwendung könnte z.B. zur Sprachaufzeichnung einen Kommentar diktieren, diesen mit einem sichtbaren Repräsentations-Objekt packen und die Grafik als Objekt in eine Textdatei einbetten. Sobald ein Benutzer die Grafik dieses Objekts (z.B. ein Lippenpaar) doppelt anklickt, würde die Anwendung zur Sprachaufzeichnung den aufgenommenen Kommentar abspielen. Verknüpfte und eingebettete Objekte eignen sich ebenso für die Implementierung wie zum Beispiel animierte Zeichnungen, ausführbare Makros, hypertextartige Verbindungen usw.

Zwischen OLE und DDEML wählen

Anwendungen können Daten austauschen, indem sie entweder die OLE oder die DDEML benutzen. Wenn eine Anwendung nicht unbedingt mehrere Elemente in einer einzelnen Konversation verwalten muß, sollte sie die OLE anstelle der DDEML verwenden.

Sowohl die OLE als auch die DDEML sind Systeme, die auf Botschaften basieren und von dynamischen Linkbibliotheken unterstützt werden. Die Entwickler sollten diese Bibliotheken anstelle der auf Botschaften basierenden Protokolle benutzen. Nähere Informationen über das OLE-Protokoll finden Sie im Abschnitt »DDE direkt anwenden«.

Anders als die OLE unterstützt die DDEML mehrere Elemente in einer Konversation. Mit der OLE muß eine Client-Anwendung, die in einem Dokument Verknüpfungen zu verschiedenen Objekten benötigt, eine separate Konversation für jedes Objekt einrichten. Die OLE bietet dagegen folgende Vorteile, die die DDEML nicht aufweist:

Erweiterbarkeit auf zukünftige Verbesserungen: Die OLE-Bibliotheken könnten in zukünftigen Versionen aktualisiert werden, um neue Datenformate, die Überwachung von Verknüpfungen, das Bearbeiten verknüpfter Objekte ohne die Client-Anwendung zu verlassen und andere Verbesserungen zu unterstützen. Diese Verbesserungen wären für Anwendungen, die die DDEML benutzen, nicht sofort verfügbar.

Dauerhaftes Einbetten und Verknüpfen der Objekte: Die OLE-Bibliothek führt die Hauptarbeit beim Aktivieren der Objekte durch, wenn ein eingebettetes Dokument geöffnet wird. Sie richtet die Konversation zwischen Client- und Server-Anwendung wieder ein. Im Gegensatz dazu liegt bei einer DDE-Verknüpfung (DDE-Advise-Schleife) die Verantwortung entweder beim Benutzer (im Falle einer vorübergehenden Verknüpfung) oder bei der Anwendung (im Falle einer dauerhaften Verknüpfung).

Übergabe allgemeiner Datenformate: Die OLE-Bibliothek übernimmt die Übergabe allgemeiner Datenformate an einen Bildschirmkontext. Eine DDE-Anwendung muß diese Arbeit selbst verrichten.

Übergabe spezieller Datenformate durch die Server-Anwendung: Die OLE- Bibliothek erleichtert die Übergabe spezieller Datenformate in den Bildschirmkontext der Client-Anwendung. Dabei führt die Server-Anwendung oder ein Objekt-Handler die eigentliche Übergabe durch. Für die Client-Anwendung ist der Arbeitsaufwand zur Übergabe der eingebetteten oder verknüpften Daten in den Bildschirmkontext sehr gering. Eine solche Übergabe von eingebetteten oder verknüpften Daten übersteigt die Fähigkeiten der DDEML.

Eingebettete und verknüpfte Objekte aktivieren: Die OLE-Bibliothek unterstützt das Aktivieren von Server-Anwendungen, um verknüpfte oder eingebettete Objekte zu bearbeiten oder Daten zu übergeben. Das Aktivieren der Server-Anwendungen zur Datenübergabe und zum Bearbeiten übersteigt die Fähigkeiten der DDEML.

Objekte und Verknüpfungen aus der Zwischenablage erzeugen: Benutzt eine Anwendung die Zwischenablage, um Verknüpfungen zu kopieren und einzufügen oder um Objekte auszutauschen, übernehmen die OLE-Bibliotheken die Hauptarbeit. Dagegen müssen DDE-Anwendungen die Windows-Funktionen für die Zwischenablage direkt aufrufen, um die oben genannten Operationen auszuführen.

Objekte und Verknüpfungen aus Dateien erzeugen: Die OLE-Bibliotheken unterstützen die Verwendung von Dateien zum Datenaustausch. Eine derartige Möglichkeit besteht im Rahmen der DDE-Protokolle nicht.

Da die OLE-Bibliotheken geschrieben wurden, bevor die DDEML verfügbar war, verwenden sie die DDE-Botschaften.

OLE für Standard-DDE-Operationen benutzen

Die OLE-Schnittstelle wurde in erster Linie für verknüpfte und eingebettete Objekte entworfen. Sie kann jedoch auch für Standard-DDE-Elemente verwendet werden, um folgende Aufgaben auszuführen:

- Initialisieren von Konversationen auf Grundlage von Anwendungs- und Themennamen oder Platzhaltern;
- Daten von einer Server-Anwendung in einem bestimmten Format anfordern;
- eine Advise-Schleife einrichten – der Server benachrichtigt den Client über Änderungen bestimmter Elemente und übergibt diese optional;
- Daten von einer Server- zu einer Client-Anwendung senden;

- Daten von einer Client-Anwendung an eine Server-Anwendung übergeben;
- DDE-Befehle mit Hilfe der Funktion *OleExecute* senden.

Eine OLE-Client-Anwendung erhält einen Zeiger auf eine *OLEOBJECT*-Struktur, die Informationen über den Klassen-, Dokument- und Elementnamen enthält. Diese Namen entsprechen folgenden DDE-Bezeichnungen:

OLE-Name	DDE-Name
Klassenname	Servicename (früher »Anwendungsname«)
Dokumentname	Themenname
Elementname	Elementname

Die Client-Anwendung kann mit der Funktion *OleCreateFromFile* ein Objekt erzeugen und die drei genannten Namen angeben. Benötigt die Client-Anwendung mehrere Elemente desselben Themas, muß sie für jeden Eintrag einen Zeiger auf eine *OLEOBJECT*-Struktur erhalten. Damit wird für jedes Element eine DDE-Konversation erzeugt.

OLE-Funktionen, die mit der *OLEOBJECT*-Struktur arbeiten, werden von der Client-Bibliothek auf folgende DDE-Botschaften abgebildet:

OLE-Funktion	DDE-Botschaft
OleExecute	*WM_DDE_EXECUTE*
OleRequestData	*WM_DDE_REQUEST*
OleSetData	*WM_DDE_POKE*

Einige OLE-Funktionen (wie zum Beispiel *OleActivate*) sind für diese Abbildung auf eine einzelne DDE-Botschaft zu komplex. Hier hängt die jeweilige DDE-Botschaft von weiteren Bedingungen ab. Benötigt eine Client-Anwendung die Funktionalität der Botschaft *WM_DDE_ADVISE* im Rahmen der OLE, muß sie die Verknüpfung mit dem Wert *olerender_format* für den Parameter *renderopt* erzeugen, das erforderliche Format angeben und die Funktion *OleGetData* zum Abrufen des Werts benutzen, wenn die Callback-Funktion die Benachrichtigung *OLE_CHANGED* erhält. Ist mehr als ein Element oder Format erforderlich, muß die Client-Anwendung für jedes Paar Element/Format eine *OLEOBJECT*-Struktur erzeugen. Diese Methode erzeugt für jede Advise-Transaktion eine eigene Konversation. Sie ist also ineffizient, wenn der Client viele solcher Konversationen benötigt.

Eine Server-Anwendung wird für die DDEML verfügbar, wenn sie durch Aufruf der Funktionen *OleRegisterServer* und *OleRegisterServerDoc* das Thema System und andere Themen verfügbar macht. Nimmt eine Client-Anwendung Verbindung auf und und fordert ein Element an, ruft die Server-Bibliothek die Funktion *GetObject* innerhalb der *OLESERVERDOCVTBL*-Struktur der Server-Anwendung auf, gefolgt von weiteren geeigneten server-implementierten Funktionen. Normalerweise ruft die Bibliothek die Funktion *GetData* in der *OLEOBJECTVTBL*-Struktur der Server-Anwendung auf. Sollte sich das Element ändern, solange das durch den Aufruf von *GetObject* reservierte Objekt noch nicht

freigegeben wurde, muß die Server-Anwendung eine Benachrichtigung senden. Dann können die OLE-Bibliotheken Daten an Client-Anwendungen senden, die *WM_DDE_ADVISE* gesendet haben.

OLE und DDEML zusammen benutzen

Einige Anwendungen benötigen neben Teilen der OLE auch die DDEML zur Unterstützung simultaner Verknüpfungen für mehrere häufig zu aktualisierende Elemente. Client-Anwendungen dieser Art können entweder die OLE-Bibliotheken verwenden, um Konversationen mit OLE-Server-Anwendungen einzurichten oder die DDEML, um Konversationen mit DDE-Server-Anwendungen einzuleiten.

Server-Anwendungen, die sowohl OLE als auch DDEML unterstützen müssen, müssen verschiedene Servicenamen (DDE-Anwendungsnamen) für OLE- und DDE-Konversationen verwenden. Andernfalls können die OLE- und DDEML-Bibliotheken nicht entscheiden, welche von ihnen antworten muß. Anwendungen ändern meist den Servicenamen der OLE-Konversation, da die anderen Anwendungen und der Benutzer selbst den Servicenamen der DDE-Konversation benutzen müssen. Der OLE-Servicename ist verborgen.

Datentransfer unter OLE

Dieser Abschnitt gibt einen kurzen Überblick, wie die verschiedenen Anwendungen Daten unter OLE gemeinsam benutzen. Einzelheiten über die Implementierung finden Sie in einem späteren Abschnitt dieses Kapitels.

Die Anwendungen benutzen drei dynamische Linkbibliotheken (DLL), um OLE zu implementieren: OLECLI.DLL, OLESVR.DLL und SHELL.DLL. OLE selbst wird von OLECLI.DLL und OLESVR.DLL unterstützt. Die Registrierungsdatenbank wird von SHELL.DLL unterstützt.

Client-Anwendungen

Eine OLE-Client-Anwendung kann OLE-Objekte entgegennehmen, auf dem Bildschirm darstellen und speichern. Dabei können die Objekte jegliche Art von Daten enthalten. Eine Client-Anwendung bezeichnet ein Objekt dadurch, daß es deutlich umrahmt oder mit sonstigen visuellen Merkmalen gekennzeichnet wird.

Server-Anwendungen

Als OLE-Server-Anwendung wird jede Anwendung bezeichnet, die ein Objekt bearbeiten kann, sobald sie von den OLE-Bibliotheken darüber benachrichtigt

wird, daß ein Benutzer einer Client-Anwendung das Objekt ausgewählt hat. Einige Server-Anwendungen können auch andere Operationen als Edit durchführen. Wird ein Objekt in einer Client-Anwendung doppelt angeklickt, startet die mit diesem Objekt verbundene Server-Anwendung. Der Benutzer bearbeitet das Objekt dann in der Server-Anwendung. Wenn die Server-Anwendung startet, ist das Fenster normalerweise so dimensioniert, daß nur das Objekt sichtbar ist. Wird ein verknüpftes Objekt doppelt angeklickt, wird die gesamte verknüpfte Datei geladen und der verknüpfte Teil ausgewählt (markiert). Bei eingebetteten Objekten wählt der Benutzer den Befehl *Aktualisieren* aus dem Menü *Datei*, um Änderungen am Objekt zu sichern. Der Befehl *Beenden* schließt die Anwendung.

Viele Anwendungen können für verknüpfte und eingebettete Objekte als Client- und als Server-Anwendungen arbeiten.

Objekt-Handler

Einige OLE-Server-Anwendungen implementieren eine weitere OLE-Bibliothek, die Objekt-Handler genannt wird. Objekt-Handler sind dynamische Linkbibliotheken, die als Vermittler zwischen Client- und Server-Anwendung fungieren. Objekt-Handler werden von den Entwicklern von Server-Anwendungen bereitgestellt, um die Ausführungsgeschwindigkeit zu erhöhen. Zum Beispiel könnte ein Objekt-Handler dazu verwendet werden, ein geändertes Objekt neu zu zeichnen, falls die Präsentationsdaten für dieses Objekt nicht von der Client-Bibliothek übergeben werden können.

Kommunikation zwischen den OLE-Bibliotheken

Soll eine Operation an einem Objekt durchgeführt werden, benutzen Client-Anwendungen Funktionen der OLE-API, um die Client-Bibliothek OLECLI.DLL zu verständigen. Diese kommuniziert ihrerseits mit Hilfe von DDE-Botschaften mit der Server-Bibliothek OLESVR.DLL, die für das Starten und Beenden der Server-Anwendung, für die Interaktion mit der Callback-Funktion der Server-Anwendung und für die Kommunikation mit der Client-Bibliothek verantwortlich ist.

Wird ein eingebettetes Objekt von einer Server-Anwendung geändert, benachrichtigt die Server-Anwendung die Server-Bibliothek von den Änderungen. Die Server-Bibliothek benachrichtigt dann die Client-Bibliothek, die ihrerseits die Client-Anwendung verständigt. Daraufhin veranlaßt die Client-Anwendung normalerweise das Neuzeichnen des eingebetteten Objekts in der Dokumentdatei. Ändert die Server-Anwendung ein verknüpftes Objekt, benachrichtigt die Server-Bibliothek die Client-Bibliothek von den Änderungen, damit das Objekt neu gezeichnet wird.

Konventionen der Zwischenablage

Der erste Schritt beim Einbetten oder Verknüpfen eines Objekts ist der Austausch von Daten durch die OLE-Client- und Server-Anwendung über die Zwischenablage. Ein Objekt, das von einer Server-Anwendung in die Zwischenablage gelegt wird, wird durch Datenformate wie Native, OwnerLink oder ObjectLink und ein Präsentationsformat repräsentiert. Die Reihenfolge, in der diese Formate in die Zwischenablage geschrieben werden, bestimmt den Objekttyp. Ist zum Beispiel das erste von der Server-Anwendung angegebene Format Native und das zweite OwnerLink, können Client-Anwendungen die Daten dazu benutzen, ein eingebettetes Objekt zu erzeugen. Ist jedoch das erste Format OwnerLink, beschreiben die Daten ein verknüpftes Objekt.

Daten im Native-Format definieren ein Objekt vollständig für eine bestimmte Server-Anwendung. Die Daten sind möglicherweise nur für diese Server-Anwendung brauchbar. Für eingebettete Objekte stellt die Client-Anwendung Speicherplatz für die Native-Daten zur Verfügung. OwnerLink-Daten bezeichnen den Besitzer eines verknüpften oder eingebetteten Objekts.

Präsentationsformate wie *CF_METAFILEPICT*, *CF_DIB* und *CF_BITMAP* ermöglichen es der Client-Bibliothek, das Objekt in einem Dokument darzustellen. Native-Daten können auch als Präsentationsformat verwendet werden, vor allem dann, wenn ein Objekt-Handler für diese Datenklasse definiert wurde. Jedoch können Native-Daten nicht in zweifacher Weise, d.h. als Datenformat und als Präsentationssformat zur Definition eines Objekts benutzt werden. Übergibt die Server-Anwendung der Zwischenablage Native- und OwnerLink-Daten, um ein eingebettetes Objekt zu beschreiben, kann sie für dieses Objekt nicht gleichzeitig Native-Daten als Präsentationsformat verwenden. Die Fähigkeit des Objekt-Handler, Native-Daten auch als Präsentationsformat zu benutzen, gründet auf der Bedeutung der Reihenfolge der Formate. Nur durch die Reihenfolge kann zwischen einem eingebetteten oder verknüpften Objekt unterschieden werden, wenn für seine Präsentation Native-Daten verwendet werden.

ObjectLink-Daten bezeichnen die Klasse und das Dokument verknüpfter Objekte sowie das Element, das die Quelle des verknüpften Objekts ist. Ist der Elementname im ObjectLink-Format *NULL*, bezieht sich die Verknüpfung auf das gesamte Server-Dokument.

Die folgende Tabelle beschreibt die Inhalte der ObjectLink-, OwnerLink- und Native-Formate der Zwischenablage:

Formatname	Inhalt
ObjectLink	Nullterminierter String für den Klassennamen, nullterminierter String für den Dokumentnamen, String für das Element mit zwei terminierenden Nullzeichen.
OwnerLink	Nullterminierter String für den Klassennamen, nullterminierter String für den Dokumentnamen, String für das Element mit zwei terminierenden Nullzeichen.

Formatname	Inhalt
Native	Byte-Folge, die nur von der Bibliothek der Server-Anwendung oder des Objekt-Handler interpretiert wird. Dieses Format braucht nur der Server-Anwendung zur Verfügung zu stehen und muß ihr das Laden und Bearbeiten des Objekts ermöglichen.

Obwohl die ObjectLink- und OwnerLink-Formate dieselben Informationen enthalten, werden sie von den OLE-Bibliotheken für verschiedene Aufgaben verwendet. So benutzen die Bibliotheken das OwnerLink-Format, um den Besitzer eines Objekts (der von der Quelle des Objekts verschieden sein kann) und das ObjectLink-Format, um die Quelle für die Objektdaten zu bezeichnen.

Der Klassenname im ObjectLink- und OwnerLink-Format ist ein eindeutiger Name für eine Klasse von Objekten, die von einer Server-Anwendung unterstützt werden. Server-Anwendungen registrieren den Klassennamen oder Namen, die sie unterstützen, in der Registrierungsdatenbank. Der Klassenname von Windows Paintbrush ist zum Beispiel »PBrush«. Eine Anwendung kann mit Hilfe des Klassennamens Informationen über eine Server-Anwendung aus der Registrierungsdatenbank lesen (weitere Informationen über Registrierungen finden Sie im Abschnitt »Registrierung«). Normalerweise ist der Dokumentname ein vollständiger Pfadname, der eine Datei mit einem Dokument bezeichnet. Der Elementname bestimmt eindeutig den Teil eines Dokuments, der als Objekt definiert ist. Die Server-Anwendungen bestimmen die Elementnamen. Dabei kann ein Elementname jeder String sein, mit dem die Server-Anwendung einen Teil eines Dokuments bezeichnet. Elementnamen dürfen keinen Schrägstrich (/) enthalten. Nachstehend sehen Sie ein Beispiel für Daten im OwnerLink- oder ObjectLink-Format.

```
Microsoft Excel Arbeitsblatt\0c:\Verzeichnis\Dokument.xls
\0R1C1:R5C3\0\0
```

Die Reihenfolge, in der verschiedene Datenformate in die Zwischenablage gelegt werden, hängt vom Typ der kopierten Daten und von den Fähigkeiten der Server-Anwendung ab. Die folgende Tabelle zeigt die Reihenfolge der Datenformate der Zwischenablage für vier verschiedene Typen ausgewählter Daten. Ein Objekt muß nicht alle verfügbaren Formate benutzen.

Quelle	Inhalte der Zwischenablage, geordnet
Eingebettetes Objekt	Native OwnerLink Picture oder anderes Präsentationsformat (optional) ObjectLink (nur enthalten, wenn die Server-Anwendung auch Verknüpfungen unterstützt)
Verknüpftes Objekt	OwnerLink Picture oder anderes Präsentationsformat (optional; für verknüpfte Objekte können dies Native-Daten sein) ObjectLink

Quelle	Inhalte der Zwischenablage, geordnet
Bilddaten	Anwendungsspezifische Formate Native OwnerLink Picture ObjectLink
Strukturierte Daten	Strukturierte Datenformate (wenn die Auswahl nur strukturierte Daten betrifft) Native OwnerLink Picture, Text, usw. ObjectLink

Bevor Daten für ein eingebettetes oder verknüpftes Objekt in die Zwischenablage kopiert werden, legt die Server-Anwendung Beschreibungen der Datenformate in die Zwischenablage. Diese Datenformate werden in der Reihenfolge ihres Informationsgehalts aufgelistet, beginnend mit dem aussagekräftigsten. Microsoft Word würde zunächst das Rich-Text-Format (RTF) und dann erst das *CF_TEXT*-Zwischenablageformat ablegen.

Wenn ein Benutzer den Befehl *Einfügen* auswählt, fragt die Client-Anwendung die Formate in der Zwischenablage ab und benutzt das erste Format, das mit dem Ziel des Objekts kompatibel ist. Da Server-Anwendungen Daten in der Reihenfolge ihres Informationsgehalts in die Zwischenablage legen, ist das erste brauchbare von der Client-Anwendung gefundene Format auch das beste. Findet die Client-Anwendung vor dem Native-Format ein anderes akzeptables Format, trägt es die Daten in das Zieldokument ein, ohne ein eingebettetes Objekt zu erzeugen. So würde zum Beispiel ein Microsoft-Word-Dokument aus Daten der Zwischenablage im RTF-Format kein eingebettetes Objekt erzeugen. Strukturierte Daten oder ein strukturiertes Dokument würden in eine Zeichenanwendung eingebettet werden. Ist das Zieldokument dagegen ein Arbeitsblatt oder ein strukturiertes Dokument, würden die Daten in das Native-Format des Zieldokuments konvertiert werden. Wenn die Client-Anwendung vor den Native- und OwnerLink-Formaten kein Datenformat akzeptieren kann, erzeugt sie mit Hilfe dieser Formate ein eingebettetes Objekt und ermittelt dann ein geeignetes Präsentationsformat. Die Zielanwendung könnte verschiedene Formate benötigen, abhängig davon, wo die Auswahl im Zieldokument abgelegt wird. So könnte das Einfügen in ein Bild oder in normalen Fließtext verschiedene Formate erfordern.

Wählt ein Benutzer den Befehl *Verknüpfung einfügen* aus dem Menü *Bearbeiten*, sucht die Client-Anwendung in der Zwischenablage nach dem ObjectLink-Format und ignoriert die Native- und OwnerLink-Formate. Das ObjectLink-Format bezeichnet die Quellklasse, das Quelldokument und das Objekt. Findet die Anwendung neben dem ObjectLink-Format ein brauchbares Präsentationsformat, erzeugt es eine OLE-Verknüpfung mit dem Quelldokument des Objekts. Wenn das ObjectLink-Format nicht verfügbar ist, sucht die Client-Anwendung nach dem Link-Format, um eine DDE-Verknüpfung zu erzeugen. Diese Art der Verknüpfung unterstützt das OLE-Protokoll nicht.

Unterstützt eine Anwendung keine OLE-Kopien von OLE-Elementen in der Zwischenablage, ignoriert sie die Native-, OwnerLink- und ObjectLink-Formate. Das Verhalten der kopierenden Anwendung ändert sich dadurch nicht.

Registrierung

Die Registrierungsdatenbank unterstützt verknüpfte und eingebettete Objekte, indem sie für das ganze System Informationen darüber bereitstellt, ob Server-Anwendungen das OLE-Protokoll unterstützen, welches die Namen der ausführbaren Dateien dieser Anwendungen sind, welche Aktionsarten für welche Objektklassen existieren und ob für eine gegebene Objektklasse eine Objekt-Handler-Bibliothek existiert.

Wenn eine Server-Anwendung installiert wird, registriert sie sich selbst in der Registrierungsdatenbank als OLE-Server-Anwendung. Die Datenbank wird von der dynamischen Linkbibliothek SHELL.DLL unterstützt. Damit sie sich selbst als OLE-Server-Anwendung registrieren kann, muß eine Server-Anwendung in der Datenbank eintragen, daß sie ein oder mehrere OLE-Protokolle unterstützt. Die einzigen Protokolle, die von der Version 1.x der Microsoft-OLE-Bibliotheken unterstützt werden, sind StdFileEditing und StdExecute. StdFileEditing ist das gebräuchlichste Protokoll für verknüpfte und eingebettete Objekte. StdExecute wird dagegen nur von Anwendungen benutzt, die die Funktion *OleExecute* unterstützen. Der dritte Name Static beschreibt ein Bild, das mit normalen OLE-Techniken nicht bearbeitet werden kann.

Wenn eine Client-Anwendung ein verknüpftes oder eingebettetes Objekt aktiviert, findet die Client-Bibliothek die Kommandozeile für die Server-Anwendung in der Datenbank, fügt die Option */Embedding* oder */Embedding Dateiname* hinzu und startet die Server-Anwendung mit der neuen Kommandozeile. Dieses Starten der Server-Anwendung unterscheidet sich wesentlich vom Starten der Anwendung durch den Benutzer. Vor dem Wort *Embedding* kann entweder ein Schrägstrich (/) oder ein Bindestrich (-) stehen. Genauere Informationen über das Verhalten einer Server-Anwendung beim Starten mit diesen Optionen finden Sie im Abschnitt »Objekte öffnen und schließen«.

Die Einträge in der Registrierungsdatenbank werden immer dann benutzt, wenn eine Anwendung oder eine Bibliothek Informationen über eine OLE-Server-Anwendung benötigt. Client-Anwendungen, die den Befehl *Objekt einfügen* unterstützen, rufen aus der Datenbank eine Liste der OLE-Server-Anwendungen ab, die neue Objekte zur Verfügung stellen können. Die Client-Anwendung ruft auch den Namen der Server-Anwendung für das Dialogfenster *Einfügen mit Formatauswahl* aus der Registrierungsdatenbank ab.

Registrierungsdatenbank

Normalerweise fügen Anwendungen mit Hilfe des Registrier-Editors (REGEDIT.EXE) von Microsoft Windows Schlüssel-/Wert-Paare in die Registrierungsdatenbank ein. Sie können dazu aber auch die Registrierfunktionen benutzen.

Die Schlüssel und Werte werden in der Registrierungsdatenbank als nullterminierte Strings gespeichert. Schlüssel werden hierarchisch strukturiert und die Namen ihrer Komponenten durch einen Backslash (\) getrennt. Für jede Klasse, die die Server-Anwendung unterstützt, sollte der Klassenname und der Pfad-

name des Server registriert werden. Der Klassenname muß identisch mit dem String sein, den die Server-Anwendung beim Aufruf der Funktion *OleRegisterServer* benutzt. Besitzt eine Klasse eine Objekt-Handler-Bibliothek, sollte sie mit dem Schlüsselwort handler registriert werden. Außerdem sollte eine Anwendung alle Aktionsarten registrieren, die von ihrer Klasse bzw. ihren Klassen unterstützt werden. Die Aktionsarten einer Anwendung müssen in sequentieller Reihenfolge angeordnet sein: die primäre Aktionsart muß den Wert 0 und die folgenden Aktionsarten müssen die Werte 1 und 2 haben, wenn ein Objekt drei Aktionsarten unterstützt.

Wie im folgenden Beispiel gezeigt wird, sollte eine Server-Anwendung bei ihrer Installation ihre Schlüssel-/Wert-Paare angeben, um für OLE-Transaktionen verfügbar zu sein. Das Beispiel zeigt die Form, in der Schlüssel-/Wert-Paare mit Hilfe des Registrier-Editors in die Datenbank geschrieben werden. Obwohl im Beispiel der Text-String manchmal über die erste Zeile hinausgeht, dürfen die einzelnen Zeilen keinen Zeilenvorschub enthalten, wenn sie in die Datenbank geschrieben werden.

```
HKEY_CLASSES_ROOT\Klassenname   =lesbare Version des Klassennamens
HKEY_CLASSES_ROOT\.ext. =Klassenname
HKEY_CLASSES_ROOT\Klassenname \protocol\StdFileEditing\server          =
   Name der ausführbaren Datei
HKEY_CLASSES_ROOT\Klassenname \protocol\StdFileEditing\handler          =
   DLL-Name
HKEY_CLASSES_ROOT\Klassenname \protocol\StdFileEditing\verb\0          =
   primäre Aktionsart
HKEY_CLASSES_ROOT\Klassenname \protocol\StdFileEditing\verb\1          =
   sekundäre Aktionsart
```

Server-Anwendungen, die die Funktion *OleExecute* unterstützen, schreiben außerdem folgende Zeile in die Datenbank:

```
HKEY_CLASSES_ROOT\Klassenname \protocol\StdExecute\server     =
   Name der ausführbaren Datei
```

Bei der Angabe der Aktionsart kann ein Et-Zeichen (&) verwendet werden, um anzuzeigen, daß das folgende Zeichen ein Tastenkürzel ist. Wird zum Beispiel »&Bearbeiten« als Aktionsart angegeben, so ist die Taste B ein Tastenkürzel.

Eine Server-Anwendung kann den vollständigen Pfadnamen ihrer ausführbaren Datei registrieren, statt nur den Dateinamen und die Argumente anzugeben. Das Registrieren nur des Dateinamens schlägt fehl, wenn die Anwendung in einem Verzeichnis installiert ist, das nicht in der PATH-Umgebungsvariablen aufgeführt ist. Eindeutiger ist deshalb die Verwendung des vollständigen Pfadnamens.

Server-Anwendungen können Datenformate registrieren, die sie aufgrund von Aufrufen der Funktion *OleSetData* annehmen oder die sie zurückgeben können, wenn eine Client-Anwendung die Funktion *OleRequestData* aufruft. Mit dieser Information können Client-Anwendungen neu erzeugte Objekte (zum Beispiel aus ausgewählten Daten in der Client-Anwendung) initialisieren oder sie können die Server-Anwendung als »Maschine« benutzen, um Daten an

eine Grafikanwendung zu senden und dafür ein neues Bild zu erhalten. Client-Anwendungen sollten nicht vom angeforderten Datenformat abhängig sein, da die Anforderungen von der Server-Anwendung zurückgewiesen werden können.

Im folgenden Beispiel ist *format* der String-Name des Formats, wie er an die Funktion *RegisterClipboardFormat* übergeben würde, oder eines der systemdefinierten Formate der Zwischenablage (z.B. *CF_METAFILEPICT*).

```
HKEY_CLASSES_ROOT\OLE\Klassenname \protocol\StdFileEditing
    \SetDataFormats [=format]
HKEY_CLASSES_ROOT\OLE\Klassenname \protocol\StdFileEditing
    \RequestDataFormats [=format]
```

Um mit früheren Anwendungen kompatibel zu sein, liest und schreibt der systemeigene Registrierservice Informationen auch im Abschnitt [embedding] der Initialisierungsdatei WIN.INI.

Im folgenden Beispiel zeigt das Schlüsselwort picture an, daß der Server Metafiles zur Übergabe von Objekten erzeugen kann.

```
[embedding]
Klassenname=Kommentar,   Textdarstellung   des   Klassennamens,
                Pfadname der Server-Anwendung/Argumente, pi
```

Versionskontrolle für Server-Anwendungen

Server-Anwendungen sollten in ihren Native-Datenformaten die Nummern ihrer Versionen speichern. Die neuen Versionen sollten fähig sein, Daten im Native-Format der alten Versionen zu bearbeiten. Dabei ist es manchmal wichtig, dem Benutzer die Möglichkeit zu geben, Daten im alten Format zu sichern, um eine Umgebung mit den Formaten der neuen und alten Versionen zu unterstützen. Außerdem können dadurch auch Anwendungen die Daten lesen, die nur das alte Format interpretieren können.

Zu einem bestimmten Zeitpunkt kann auf einem Arbeitsplatz für einen bestimmten Klassennamen nur eine Anwendung als Server-Anwendung registriert sein. Der Klassenname (der mit den Native-Daten für Objekte gespeichert wird) und die Server-Anwendung sind in der Registrierungsdatenbank verknüpft, wenn sich die Server-Anwendung während der Installation registrieren läßt.

Läßt die neue Version eines Server die Weiterbenutzung der alten Version zu, sollte der neuen Server-Anwendung ein neuer Klassenname zugeteilt werden. Am besten wird dafür eine Versionsnummer an den Klassennamen gehängt. So kann der Benutzer leicht zwischen beiden Versionen unterscheiden. Die OLE-Bibliotheken überprüfen diese Nummern nicht.

Wenn eine neue Version der Server-Anwendung installiert wird, sollte der Benutzer entweder die Möglichkeit erhalten, die alten Objekte auf die neue Server-Anwendung abzubilden (dabei wird die neue Server-Anwendung als Server für beide Klassennamen registriert) oder sie getrennt von ihr aufzubewahren. Werden die alten Objekte getrennt aufbewahrt, verfügt der Benutzer über zwei Objektarten (zum Beispiel *Graph1* und *Graph2*).

Der Benutzer sollte die Möglichkeit haben, die alte Server-Version zu verwerfen, indem er die Registrierungsdatenbank mit Hilfe des Setup-Programms

aktualisiert. Beim Aktualisieren der Datenbank erhalten die alten und neuen Objekte denselben Wert für die lesbare Version des Klassennamens, obwohl ihre eigentlichen Klassennamen verschieden bleiben. Die OLE-Client-Bibliothek entfernt doppelt vorkommende Namen, wenn sie die Liste im Dialogfenster *Objekt einfügen* erzeugt. Wenn eine Client-Anwendung eine Liste durch Aufzählung der Datenbank erzeugt, muß sie selbst die doppelt vorkommenden Namen filtern.

Die Benutzerschnittstelle der Client-Anwendung

Öffnet ein Benutzer ein Dokument, das ein verknüpftes oder ein eingebettetes Objekt enthält, benutzt die Client-Anwendung OLE-Funktionen, um mit OLECLI.DLL zu kommunizieren. Diese Bibliothek unterstützt die Client-Anwendung unter anderem beim Laden, Zeichnen und Aktualisieren von Objekten. Außerdem hilft sie der Client-Anwendung bei der Interaktion mit Server-Anwendungen.

Neue und geänderte Befehle

Normalerweise implementiert eine OLE-Client-Anwendung die folgenden neuen oder geänderten Befehle als Teil ihres Menüs *Bearbeiten*. Obwohl diese Benutzerschnittstelle nicht notwendig ist, wird sie für die Konsistenz mit bestehenden OLE-Anwendungen empfohlen.

Befehl	Beschreibung
Kopieren	Kopiert ein Objekt aus einem Dokument in die Zwischenablage.
Ausschneiden	Entfernt ein Objekt aus einem Dokument und legt es in die Zwischenablage.
Einfügen	Kopiert ein Objekt aus der Zwischenablage in ein Dokument.
Verknüpfung einfügen	Fügt eine Verknüpfung zwischen einem Dokument und der Datei ein, die ein Objekt enthält.
<Klassenname> Objekt	Ermöglicht es dem Benutzer, die Aktionsarten für ein verknüpftes oder eingebettetes Objekt zu aktivieren. Der tatsächlich anstelle des Platzhalters *<Klassenname>* verwendete Text hängt vom ausgewählten Objekt ab.
Verknüpfungen	Ermöglicht es dem Benutzer, die Verknüpfungsaktualisierung zu ändern, verknüpfte Objekte zu aktualisieren, Verknüpfungen aufzulösen, unterbrochene Verknüpfungen wiederherzustellen und die mit den verknüpften Objekten verbundenen Aktionsarten zu aktivieren.
Objekt einfügen	Startet die vom Benutzer in einem Dialogfenster ausgewählte Server-Anwendung und bettet das von der Server-Anwendung erzeugte Objekt in ein Dokument ein. Dieser Befehl ist optional.

Befehl	Beschreibung
Einfügen mit Formatauswahl	Überträgt ein Objekt aus der Zwischenablage in ein Dokument oder fügt eine Verknüpfung mit dem Objekt ein, indem die vom Benutzer im Dialogfenster ausgewählten Datenformate benutzt werden. Dieser Befehl ist optional.

Zusätzlich zu den aufgeführten Menüänderungen müssen die Client-Anwendungen Änderungen ihrer *Kopieren*- und *Ausschneiden*-Befehle implementieren. Wird ein verknüpftes oder eingebettetes Objekt in der Client-Anwendung ausgewählt, kann die Anwendung mit der Funktion *OleCopyToClipboard* die *Ausschneiden*- und *Kopieren*-Befehle implementieren.

Wählt der Benutzer den Befehl *Einfügen*, sollte die Client-Anwendung den Inhalt der Zwischenablage an der aktuellen Position im Dokument einfügen. Wenn die Zwischenablage ein Objekt enthält, wird mit diesem Befehl das Objekt in das Dokument eingebettet.

Wählt der Benutzer den Befehl *Verknüpfung einfügen*, dann fügt die Client-Bibliothek ein verknüpftes Objekt an der aktuellen Position im Dokument ein. Dabei wird das Objekt im Dokument dargestellt. Die Native-Daten, die dieses Objekt definieren, werden anderswo gespeichert.

Wenn ein Benutzer ein verknüpftes Objekt in die Zwischenablage kopiert, können andere Dokumente dieses Objekt benutzen, um eine Verknüpfung mit den Native-Daten zu erzeugen.

Mit dem Befehl *<Klassenname>Objekt* kann der Benutzer eine der Aktionsarten eines Objekts wählen. Wenn die Auswahl im Dokument ein eingebettetes Objekt ist, wird der Platzhalter *<Klassenname>* normalerweise durch die Klasse und den Namen des Objekts ersetzt. Wählt ein Benutzer zum Beispiel ein Objekt aus, das aus einem Zellenbereich eines Arbeitsblatts von Microsoft Excel besteht, kann der Befehl »Microsoft Excel Arbeitsblatt Objekt« lauten. Falls ein Objekt nur eine Aktionsart unterstützt, sollte im Menüeintrag der Name der Aktionsart hinter »Objekt« stehen. So könnte der Befehl zum Beispiel »WPDokument Objekt bearbeiten« lauten, wenn die einzige Aktionsart eines Textobjekts *Bearbeiten* ist. Unterstützt ein Objekt mehr als eine Aktionsart, führt die Auswahl des Befehls *<Klassenname> Objekt* zur Anzeige einer Liste der verfügbaren Aktionsarten. Weitere Informationen über Aktionsarten finden Sie im Abschnitt »Aktionsarten«.

Die Auswahl des Befehls *Verknüpfungen* öffnet das Dialogfenster *Verknüpfungen*, das die ausgewählten Verknüpfungen mit ihren Quelldokumenten auflistet und es dem Benutzer ermöglicht, die Art der Aktualisierung der Verknüpfungen zu bestimmen. Der Benutzer kann mit diesem Dialogfenster Verknüpfungen mit Objekten »reparieren«, die entfernt oder umbenannt worden sind.

Bei Auswahl des Befehls *Einfügen mit Formatauswahl* sollte eine Client-Anwendung ein Dialogfenster öffnen, in dem die von der Client-Anwendung unterstützten Datenformate aufgelistet sind, die gegenwärtig in der Zwischenablage zur Verfügung stehen. Im Dialogfenster *Einfügen mit Formatauswahl* kann der Benutzer das voreingestellte Verhalten der Befehle *Einfügen* und *Verknüpfungen* überschreiben.

Kann die Client-Anwendung beispielsweise das erste Format in der Zwischen-

ablage bearbeiten, besteht das Verhalten der Client-Anwendung darin, die Daten in das Dokument zu kopieren, ohne sie in ein Objekt zu schreiben. Mit Hilfe des Befehls *Einfügen mit Formatauswahl* kann der Benutzer dieses Verhalten überschreiben und aus den Daten ein Objekt erzeugen.

Bei Auswahl des Befehls *Objekt einfügen* sollte eine Client-Anwendung dem Benutzer ermöglichen, ein Objekt der angegebenen Klasse an der aktuellen Position in ein Dokument einzufügen. Um einen Bereich von Zellen eines Arbeitsblatts in ein Textdokument einzufügen, könnte der Benutzer den Befehl *Objekt einfügen* wählen und im Dialogfenster den Eintrag *Microsoft Excel Arbeitsblatt* auswählen, um Microsoft Excel zu starten. Der Benutzer würde mit Microsoft Excel ein Objekt erzeugen, das in das Textdokument eingebettet werden soll und anschließend Microsoft Excel wieder beenden. Der Zellenbereich würde automatisch in das Textdokument eingefügt.

Der Befehl *Objekt einfügen* ist optional, da ein Benutzer mit einer anderen, weniger komfortablen Prozedur dasselbe Ergebnis erzielen könnte. Gemäß dem Beispiel im vorhergehenden Absatz könnte der Benutzer die Client-Anwendung verlassen, Microsoft Excel starten und die Daten mit dem Befehl *Ausschneiden* oder *Kopieren* in die Zwischenablage übertragen. Nachdem der Benutzer zur Client-Anwendung zurückgekehrt ist, könnte er mit dem Befehl *Einfügen* die Daten aus der Zwischenablage in das Textdokument schreiben.

Nachdem der Benutzer ein Objekt aktiviert hat, kann er mit dem Befehl *Rückgängig* alle Änderungen verwerfen, die seit der letzten Aktualisierung bzw. Aktivierung des Objekts durchgeführt wurden. Das Objekt erhält wieder den Zustand, den es vor der Aktualisierung hatte. Der Befehl *Rückgängig* bricht außerdem die Verbindung mit der Server-Anwendung ab.

Pakete verwenden

Unter einem Paket versteht man ein eingebettetes Grafikobjekt, das ein weiteres (verknüpftes oder eingebettetes) Objekt enthält. So kann z.B. ein Benutzer eine Datei in ein Symbol packen und dieses in ein OLE-Dokument einbetten. Die meisten Fähigkeiten zum Erstellen von Paketen werden von der dynamischen Linkbibliothek SHELL.DLL bereitgestellt.

Ein Benutzer kann ein Paket auf verschiedene Arten in ein OLE-Dokument einbinden:

- Kopieren einer Datei aus dem Datei-Manager in die Zwischenablage und Auswahl des Befehls *Einfügen* oder *Verknüpfung einfügen* aus dem Menü *Bearbeiten* der Client-Anwendung;

- ziehen einer Datei aus dem Datei-Manager in das offene Fenster eines Dokuments in einer Client-Anwendung;

- auswählen von *Paket* aus der Objektliste im Dialogfenster *Objekt einfügen*. Dadurch wird der Objekt-Manager gestartet, mit dem der Benutzer eine Datei oder Daten mit einem Symbol oder einer Grafik verbinden kann. Durch Auswahl von *Aktualisieren* und *Beenden* aus dem Menü *Datei* des Objekt-Managers wird das Paket in das Client-Dokument eingebunden;

- direktes Starten des Objekt-Managers, gefolgt von den im vorherigen Absatz angegebenen Schritten.

Ein Benutzer, dessen System nicht über den Datei-Manager der Windows-Version 3.1 verfügt, kann mit folgenden Schritten ein Paket mit dem Objekt-Manager erzeugen:

- Kopieren der zu packenden Daten in die Zwischenablage.
- Öffnen des Objekt-Managers und Einfügen der Daten (hier kann der Benutzer das voreingestellte Symbol und dessen Beschriftung ändern).
- Auswahl von *Paket kopieren* im Menü *Bearbeiten* des Objekt-Managers, um das Paket in die Zwischenablage zu kopieren.
- Auswahl des Befehls *Einfügen* im Menü *Bearbeiten* der Client-Anwendung, um das Paket einzubetten.

Die Benutzerschnittstelle der Server-Anwendung

Eine Server-Anwendung für verknüpfte und eingebettete Objekte ist jede Anwendung, mit der ein Objekt bearbeitet werden kann, nachdem ihr von den OLE-Bibliotheken mitgeteilt wurde, daß der Benutzer einer Client-Anwendung das Objekt aktiviert hat (einige Server-Anwendungen verwenden andere Aktionsarten als »Edit«, um mit einem Objekt zu arbeiten). Obwohl Client-Anwendungen viele Änderungen in der Benutzerschnittstelle implementieren, um OLE zu unterstützen, ändert sich die Benutzerschnittstelle für Server-Anwendungen kaum.

Normalerweise implementieren OLE-Server-Anwendungen Änderungen in den folgenden Befehlen des Menüs *Bearbeiten*. Obwohl diese Benutzerschnittstelle nicht verbindlich ist, wird sie für die Konsistenz mit bereits bestehenden OLE-Anwendungen empfohlen.

Befehl	Beschreibung
Ausschneiden	Überträgt Daten aus der Anwendung in die Zwischenablage und löscht die Daten im Quelldokument. Eine Client-Anwendung kann mit diesen Daten ein eingebettetes Objekt erzeugen.
Kopieren	Überträgt eine Kopie der Daten aus der Anwendung in die Zwischenablage. Eine Client-Anwendung kann mit diesen Daten ein eingebettetes Objekt erzeugen und möglicherweise eine Verknüpfung mit dem Quelldokument einrichten.

Wenn eine Server-Anwendung im Rahmen der Objektaktivierung aus einem Verbunddokument gestartet wird, ändern einige Menüeinträge ihre Namen oder verhalten sich anders. Das genaue Verhalten der Server-Anwendung hängt davon ab, ob MDI unterstützt wird.

Objekte eines Mehrfachinstanzen-Server aktualisieren

Unter einem Mehrfachinstanzen-Server versteht man eine Server-Anwendung, die MDI nicht unterstützt. Wird ein eingebettetes Objekt von einer

solchen Server-Anwendung bearbeitet oder abgespielt, sollte der Befehl *Speichern* im Menü *Datei* in *Aktualisieren* geändert werden. Wenn eine Server-Anwendung für ein verknüpftes Objekt gestartet wird, sollte diese Änderung nicht vorgenommen werden. Wenn der Benutzer den Befehl *Aktualisieren* auswählt, wird das Objekt in der Client-Anwendung aktualisiert. Der Fokus bleibt jedoch beim Server-Fenster, das mit dem Befehl *Beenden* geschlossen werden kann.

Bei Auswahl der Befehle *Speichern unter*, *Neu* oder *Öffnen* sollte die Anwendung zunächst eine Warnmeldung anzeigen und rückfragen, ob das Objekt im Verbunddokument aktualisiert werden soll, bevor der Befehl ausgeführt wird. Die Befehle *Neu* und *Öffnen* beenden die Verbindung zwischen der Client- und der Server-Anwendung. Dies ist auch bei Auswahl des Befehls *Speichern unter* der Fall, wenn die Server-Anwendung ein eingebettes Objekt bearbeitet hat.

Objekte eines Einfachinstanzen-Server aktualisieren

Für das Aktualisieren von Objekten aus Einfachinstanzen (MDI)-Server-Anwendungen gelten die gleichen Regeln wie für das Aktualisieren von Objekten aus Mehrfachinstanzen-Server-Anwendungen, mit folgenden Ausnahmen:

- Wechselt der Fokus in einer MDI-Server-Anwendung von einem Fenster, in dem ein eingebettetes Objekt aktiviert wurde, zu einem Fenster, in dem ein Dokument bearbeitet wird, das kein eingebettetes Objekt enthält, sollte der Befehl *Aktualisieren* wieder in den Befehl *Speichern* geändert werden.

- Wählt der Benutzer den Befehl *Neu* oder *Öffnen*, so bleibt das Fenster mit dem eingebetteten Objekt geöffnet. Dadurch entfällt die Notwendigkeit der Benutzeranfrage, ob das Objekt aktualisiert werden soll.

Speicherformate für Objekte

Die Präsentationsdaten in verknüpften und eingebetteten Objekten können als Präsentationsobjekt angesehen werden. Ein Präsentationsobjekt kann ein Standard-, ein generisches oder ein *NULL*-Objekt sein. Ist das Format vom Typ Metafile, Bitmap oder geräteunabhängiges Bitmap (DIB = device-independent bitmap), so wird ein Standard-Präsentationsobjekt verwendet. Die Client-Bibliothek unterstützt die Präsentationsobjekte und deren Anzeige. Weder Client-Anwendungen noch Objekt-Handler können die Präsentationsobjekte benutzen, da diese ausschließlich der Client-Bibliothek zur Verfügung stehen.

Die folgende Tabelle gibt das Speicherformat für Strings in OLE an. Die Elemente werden in der angegebenen Reihenfolge gespeichert.

Typ	Beschreibung
LONG	Länge des String, einschließlich des abschließenden Null-Zeichens.
Variabel	Nullterminierte Byte-Folge.

Die folgende Tabelle enthält das Speicherformat für Standard-Präsentationsobjekte, die für verknüpfte und eingebettete Objekte benutzt werden. Die Elemente werden in der angegebenen Reihenfolge gespeichert.

Typ	Beschreibung
LONG	OLE-Versionsnummer.
LONG	Formatbezeichner. Dieser Wert ist 5.
Variabel	Klassen-String. Für Standard-Präsentationsobjekte ist dieser String *METAFILEPICT*, *BITMAP* oder *DIB*.
LONG	Breite des Objekts in *MM_HIMETRIC*-Einheiten.
LONG	Höhe des Objekts in *MM_HIMETRIC*-Einheiten.
LONG	Größe der Präsentationsdaten in Byte.
Variabel	Präsentationsdaten

Die folgende Tabelle enthält das Speicherformat für das generische Präsentations-Objekt, das für verknüpfte und eingebettete Objekte verwendet wird. Generische Objekte werden benutzt, wenn das Format der Zwischenablage nicht Metafile, Bitmap oder DIB ist. Die Elemente werden in der angegebenen Reihenfolge gespeichert.

Typ	Beschreibung
LONG	OLE-Versionsnummer.
LONG	Formatbezeichner. Dieser Wert ist 5.
Variabel	Klassen-String.
LONG	Wert des Zwischenablageformats. Existiert dieser Wert, folgt als nächster Speichereintrag die Größe der Präsentationsdaten.
LONG	Name des Zwischenablageformats. Dieser Wert existiert nur, wenn der Wert des Zwischenablageformats *NULL* ist.
LONG	Größe der Präsentationsdaten in Byte.
Variabel	Präsentationsdaten.

Die folgende Tabelle gibt das Speicherformat für eingebettete Objekte an. Die Elemente werden in der angegebenen Reihenfolge gespeichert.

Typ	Beschreibung
LONG	OLE-Versionsnummer
LONG	Formatbezeichner. Dieser Wert ist 2.
Variabel	Klassen-String.
Variabel	Themen-String.
Variabel	Element-String.
LONG	Größe der Native-Daten in Byte.
Variabel	Native-Daten.
Variabel	Präsentationsdaten (Standard, generisch oder *NULL*).

Die folgende Tabelle gibt das Speicherformat für verknüpfte Objekte an. Die Elemente werden in der angegebenen Reihenfolge gespeichert.

Typ	Beschreibung
LONG	OLE-Versionsnummer.
LONG	Formatbezeichner. Dieser Wert ist 1.
Variabel	Klassen-String.
Variabel	Themen-String.
Variabel	Element-String
Variabel	String für den Netznamen.
short	Netztyp.
short	Versionsnummer des Netztreibers.
LONG	Optionen zur Aktualisierung der Verknüpfung.
Variabel	Präsentationsobjekt (Standard, generisch oder *NULL*).

Die folgende Tabelle gibt das Speicherformat für statische Objekte an. Der einzige Unterschied zu dem Format für Standard-Präsentationsobjekte besteht im Wert des Formatbezeichners. Die Elemente werden in der angegebenen Reihenfolge gespeichert.

Typ	Beschreibung
LONG	OLE-Versionsnummer.
LONG	Formatbezeichner. Dieser Wert ist 3.
Variabel	Klassen-String. Für statische Objekte ist dieser String *METAFILEPICT*, *BITMAP* oder *DIB*.
LONG	Breite des Objekts in *MM_HIMETRIC*-Einheiten.
LONG	Höhe des Objekts in *MM_HIMETRIC*-Einheiten.
LONG	Größe der Präsentationsdaten in Byte.
Variabel	Präsentationsdaten.

Client-Anwendungen

Eine Client-Anwendung benutzt eine Server-Anwendung, um ein in einem Verbunddokument enthaltenes Objekt zu aktivieren und zu übergeben. Die Client-Anwendung stellt Speicherplatz für eingebettete Objekte, Kontextinformationen (Zieldrucker, Seitenpositionierung u.ä.) und die Mittel für den Benutzer zur Aktivierung des Objekts und die mit dem Objekt verbundene Server-Anwendung zur Verfügung. Client-Anwendungen ermöglichen es auch, eingebettete und verknüpfte Objekte in einem Dokument abzulegen und diese wieder aus dem Dokument zu entfernen.

Client-Anwendungen müssen permanenten Speicherplatz für Objekte in der Datei des Verbunddokuments bereitstellen. Ist das gesicherte Element ein eingebettetes Objekt, werden die Native-Daten des Objekts, die Präsentationsdaten für das Objekt (zum Beispiel ein Metafile) und die OwnerLink-Informationen von der Bibliothek gespeichert. Falls das zu sichernde Element eine Verknüpfung mit einem anderen Dokument ist, speichert die Client-Bibliothek die Präsentationsdaten und das ObjectLink-Format.

Client-Anwendungen unterstützen asynchrone Operationen, indem sie Callback-Funktionen definieren, an die die Bibliothek Benachrichtigungen über die laufenden Operationen sendet. Solange die Client-Anwendung eingehende Botschaften verarbeitet, kann sie auf Benachrichtigungen der Callback-Funktion und auf Eingaben des Benutzers reagieren.

Weitere Informationen zu asynchronen Operationen finden Sie im Abschnitt »Asynchrone Operationen«.

Eine Client-Anwendung starten

Der Start einer Client-Anwendung sollte folgendermaßen stattfinden:

1. Registrierung der benötigten Zwischenablageformate.
2. Reservierung und Initialisierung der benötigten *OLECLIENT*-Strukturen.
3. Reservierung und Initialisierung einer *OLESTREAM*-Struktur.

Eine Client-Anwendung kann die Zwischenablageformate registrieren, indem die Funktion *RegisterClipboardFormat* für jedes benötigte Format aufgerufen wird: zum Beispiel Native, OwnerLink, ObjectLink und weitere benötigte Formate. Eine Client-Anwendung verwendet zwei verschiedene Strukturen, um Informationen von der Client-Bibliothek zu erhalten: *OLECLIENT* und *OLESTREAM*.

Die *OLECLIENT*-Struktur zeigt auf eine *OLECLIENTVTBL*-Struktur, die ihrerseits auf eine Callback-Funktion zeigt, die durch die Client-Anwendung bereitgestellt wird. Die OLE-Bibliotheken benutzen diese Callback-Funktion, um die Client-Anwendung über Änderungen an einem Objekt zu benachrichtigen. Die Parameter der Callback-Funktion sind ein Zeiger auf eine Client-Struktur, ein Zeiger auf das jeweilige Objekt und ein Wert, der den Grund der Benachrichtigung angibt. Normalerweise erzeugt eine Anwendung eine *OLECLIENT*-Struktur für jede *OLEOBJECT*-Struktur. Besitzt eine Anwendung für jedes Objekt eine eigene *OLECLIENT*-Struktur, so kann eine Anwendung als Reaktion auf die Callback-Benachrichtigung *OLE_QUERY_PAINT* objektspezifische Operationen durchführen.

Die *OLECLIENT*-Struktur kann ebenso auf Daten zeigen, die den Status eines Objekts beschreiben. Diese Daten, falls vorhanden, werden nur von der Client-Anwendung bereitgestellt und benutzt. Die Client-Anwendung reserviert eine eigene *OLECLIENT*-Struktur für jedes Objekt und speichert darin die Statusinformationen des Objekts. Da ein Argument der Callback-Funktion ein Zeiger auf die *OLECLIENT*-Struktur ist, ist der Aufruf der Callback-Funktion eine effiziente Methode, um Statusinformationen über das Objekt abzurufen.

Die *OLESTREAM*-Struktur zeigt auf eine *OLESTREAMVTBL*-Struktur, die weitere Zeiger enthält. Die Zeiger in dieser Tabelle zeigen auf Funktionen, die von der Client-Anwendung bereitgestellt werden, um Ein- und Ausgaben in einen Stream durchzuführen. Die Client-Bibliotheken benutzen diese Funktionen beim Laden und Speichern von Objekten. Eine Client-Anwendung kann Funktionen für bestimmte Situationen anpassen und den permanenten Speicher-

platz für ein Objekt verändern. Eine Client-Anwendung könnte z.B. ein Objekt in einer Datenbank anstelle einer Datei mit dem Rest des Dokuments abspeichern.

Die Client-Anwendung sollte mit der Funktion *MakeProcInstance* Zeiger auf die Callback-Funktion in der *OLECLIENTVTBL*-Struktur und auf die Funktionen in der *OLESTREAMVTBL*-Struktur erzeugen. Callback-Funktionen sollten in der Moduldefinitionsdatei exportiert werden.

Ein Verbund-dokument öffnen

Um ein Verbunddokument zu öffnen, sollte eine Client-Anwendung folgende Schritte ausführen:

1. Das Dokument für die Client-Bibliothek registrieren.
2. Die Daten des Dokuments aus einer Datei laden.
3. Für jedes Objekt im Dokument die Funktion *OleLoadFromStream* aufrufen.
4. Jedes Objekt auflisten, das manuell verknüpft ist, so daß der Benutzer diese Objekte aktualisieren kann. Alle automatischen Verknüpfungen sollten automatisch aktualisiert werden.

Die Funktion *OleRegisterClientDoc* registriert ein Dokument für die Client-Bibliothek und liefert ein Handle zurück, das in Funktionen benutzt wird, die Objekte erzeugen und Dokumente verwalten. Diese Registrierung bezieht sich nicht auf die Registrierungsdatenbank.

Eine Client-Anwendung sollte für jedes Objekt im Dokument die Funktion *OleLoadFromStream* aufrufen, das auf dem Bildschirm angezeigt oder anderweitig aktiviert wird. Es ist meistens nicht notwendig, jedes Objekt in einem Dokument sofort nach dem Öffnen des Dokuments zu laden. Zu den Parametern dieser Funktion zählt unter anderem ein Zeiger auf eine *OLECLIENT*-Struktur, die zum Auffinden der richtigen Callback-Funktion der Client-Anwendung benutzt wird (und die manchmal von der Client-Anwendung benutzt wird, um private Statusinformationen über das Objekt abzuspeichern). Ein weiterer Parameter der Funktion *OleLoadFromStream* ist ein Zeiger auf die *OLESTREAM*-Struktur. Die Bibliothek ruft die Funktion *Get* der *OLESTREAMVTBL*-Struktur auf, um das Objekt zu laden.

Dokumente verwalten

Eine Client-Anwendung sollte die Bibliothek benachrichtigen, wenn sie ein Dokument öffnet, schließt, sichert, umbenennt oder einen vorher gesicherten Zustand eines Dokuments wiederherstellt.

Eine Client-Anwendung kann folgende Funktionen benutzen, um diese Aufgaben durchzuführen.

Funktion	Beschreibung
OleRegisterClientDoc	Registriert ein geöffnetes Dokument für die Bibliothek.
OleRenameClientDoc	Informiert die Bibliothek über eine Umbenennung eines Dokuments.
OleRevertClientDoc	Informiert die Bibliothek darüber, daß ein Dokument in einem zuvor gesicherten Zustand wiederhergestellt worden ist.
OleRevokeClientDoc	Informiert die Bibliothek darüber, daß ein Dokument geschlossen werden sollte oder nicht mehr existiert.
OleSavedClientDoc	Informiert die Bibliothek darüber, daß ein Dokument gespeichert worden ist.

Eine Client-Anwendung sollte einen feststehenden Namen für jedes Objekt verwalten. Dieser Name sollte innerhalb des Client-Dokuments eindeutig sein und zusammen mit dem Dokument abgespeichert werden. Der Name wird beim Öffnen des Dokuments angegeben und sollte sich auch dann nicht ändern, wenn das Dokument gesichert oder erneut geöffnet wird. Benutzt eine Client-Anwendung die Funktion *OleRename*, um den Namen eines Objekts zu ändern, darf der neue Name im Dokument noch nicht vorhanden sein und muß zusammen mit dem Dokument abgespeichert werden.

Ein Dokument speichern

Eine Client-Anwendung sollte folgende Schritte ausführen, um ein Dokument zu speichern:

1. Die Daten des Dokuments in der Dokumentdatei abspeichern.

2. Für jedes Objekt im Dokument die Funktion *OleSaveToStream* aufrufen.

3. Sobald die Bibliothek bestätigt, daß alle Objekte gespeichert worden sind, die Funktion *OleSavedClientDoc* aufrufen.

Bevor eine Client-Anwendung die Funktion *OleSaveToStream* aufruft, kann sie mit der Funktion *OleQuerySize* die für das Abspeichern eines Objekts benötigte Puffergröße bestimmen.

Ein Dokument schließen

Eine Client-Anwendung sollte folgende Schritte ausführen, um ein Dokument zu schließen:

1. Für jedes Objekt im Dokument die Funktion *OleRelease* aufrufen.

2. Eine der Funktionen *OleRevertClientDoc* oder *OleSavedClientDoc* aufrufen, um den aktuellen Status des Dokuments für die Bibliothek zu registrieren.

3. Sobald die Bibliothek bestätigt hat, daß alle Objekte geschlossen worden sind, die Funktion *OleRevokeClientDoc* aufrufen.

Asynchrone Operationen

Ruft eine Client-Anwendung eine Funktion auf, die ihrerseits eine Server-Anwendung startet, können die Aktionen von Client- und Server-Anwendung asynchron verlaufen. Beispielsweise verläuft das Aktualisieren eines Dokuments und das Schließen einer Server-Anwendung asynchron. Sobald eine asynchrone Operation beginnt, liefert die Client-Bibliothek *OLE_WAIT_FOR_RELEASE* zurück. Erhält eine Client-Anwendung diese Benachrichtigung, muß sie auf die Benachrichtigung *OLE_RELEASE* warten, bevor sie beendet wird. Falls die Client-Anwendung keine weiteren Aktionen vornehmen kann, bis die asynchrone Operation abgeschlossen ist, sollte sie in eine modale Botschaftsschleife eintreten und auf *OLE_RELEASE* warten. Andernfalls sollte die Client-Anwendung der Hauptbotschaftsschleife die normale Weiterbearbeitung von Botschaften ermöglichen.

Eine Anwendung kann nur eine asynchrone Operation zur gleichen Zeit für ein Objekt ausführen. Vor jeder weiteren asynchronen Operation muß für das jeweilige Objekt die Botschaft *OLE_RELEASE* empfangen werden. Die Callback-Funktion der Client-Anwendung muß für alle laufenden asynchronen Operationen die Botschaft *OLE_RELEASE* erhalten, bevor die Funktion *OleRevokeClientDoc* aufgerufen werden kann.

Einige der Funktionen zur Erzeugung von Objekten liefern *OLE_WAIT_FOR_RELEASE* zurück. Die Client-Anwendung kann mit der Arbeit am Dokument fortfahren, während sie auf *OLE_RELEASE* wartet. Einige Funktionen (z.B. *OleActivate*) können aber solange nicht aufgerufen werden, bis die asynchrone Operation abgeschlossen ist.

Ruft eine Anwendung eine Funktion für ein Objekt auf, bevor die Botschaft *OLE_RELEASE* für dieses Objekt empfangen wird, kann die Funktion *OLE_BUSY* zurückliefern. Auch die Server-Anwendung liefert *OLE_BUSY* zurück, wenn sich die Bearbeitung einer neuen Anforderung und der Bearbeitung der aktuellen Anforderung einer Client-Anwendung oder des Benutzers überschneidet. Liefert eine Funktion *OLE_BUSY* zurück, kann die Client-Anwendung eine entsprechende Meldung anzeigen oder in einer Schleife auf *OLE_OK* warten. Wenn die Server-Anwendung beschäftigt ist, wird außerdem der Callback-Funktion der Anwendung die Benachrichtigung *OLE_QUERY_RETRY* übergeben. Liefert die Callback-Funktion *FALSE*, wird die Transaktion mit der Server-Anwendung beendet. Falls die Server-Anwendung die Funktion *OleBlockServer* verwendet, um OLE-Aktivitäten später zu bearbeiten, wird der Client-Anwendung die Benachrichtigung *OLE_QUERY_RETRY* nicht übergeben.

Das folgende Beispiel zeigt eine Botschaftsschleife, die es der Client-Anwendung ermöglicht, Botschaften zu bearbeiten, während sie auf die Benachrichtigung *OLE_RELEASE* wartet:

```
while ((olestat = OleQueryReleaseStatus(lpObject)) == OLE_BUSY) {
   if (GetMessage(&msg, NULL, NULL, NULL)) {
       TranslateMessage(&msg);
       DispatchMessage(&msg);
   }
}
if (olestat == OLE_ERROR_OBJECT) {
   .
   . /* Der Parameter lpObject ist ungültig. */
   .
}
else { /* if olestat == OLE_OK */
   .
   . /* Objekt wurde freigegeben oder die Server-Anwendung
   .    beendet. */
}
```

Eine Server-Anwendung könnte unerwartet abgebrochen werden, während die Client-Anwendung noch auf *OLE_RELEASE* wartet. In diesem Fall kann die Client-Bibliothek nur dann wieder richtig einsetzen, wenn die Client-Anwendung die Funktion *OleQueryReleaseStatus* benutzt (siehe vorhergehendes Beispiel).

Die folgende Tabelle zeigt, welche OLE-Funktionen die Botschaften *OLE_WAIT_FOR_RELEASE* oder *OLE_BUSY* an eine Client-Anwendung zurückliefern können:

Funktion	*OLE_BUSY*	*OLE_WAIT_FOR_RELEASE*
OleActivate	Ja	Ja
OleClose	Ja	Ja
OleCopyFromLink	Ja	Ja
OleCreate	Nein	Ja
OleCreateFromClip	Nein	Ja
OleCreateFromFile	Nein	Ja
OleCreateFromTemplate	Nein	Ja
OleCreateLinkFromClip	Nein	Ja
OleCreateLinkFromFile	Nein	Ja
OleDelete	Ja	Ja
OleExecute	Ja	Ja
OleLoadFromStream	Nein	Ja
OleObjectConvert	Ja	Nein
OleReconnect	Ja	Ja
OleRelease	Ja	Ja
OleRequestData	Ja	Ja
OleSetBounds	Ja	Ja
OleSetColorScheme	Ja	Ja
OleSetData	Ja	Ja
OleSetHostNames	Ja	Ja
OleSetLinkUpdateOptions	Ja	Ja

Funktion	*OLE_BUSY*	*OLE_WAIT_FOR_RELEASE*
OleSetTargetDevice	Ja	Ja
OleUnlockServer	Nein	Ja
OleUpdate	Ja	Ja

Objekte anzeigen und drucken

Wurde ein Objekt geladen und – falls nötig – aktualisiert, kann das Objekt mit dem Verbunddokument am Bildschirm dargestellt oder ausgedruckt werden. Um ein Objekt anzuzeigen, sollte die Client-Anwendung den Gerätekontext und das umgebende Rechteck initialisieren und sicherstellen, daß derselbe Abbildungsmodus verwendet wird. Anschließend sollte sie die Funktion *OleDraw* aufrufen. Die Client-Anwendung kann die Funktion *OleQueryBounds* verwenden, um die Größe des umgebenden Rechtecks auf dem Zielgerät abzurufen.

Ein Objekt-Handler kann verwendet werden, um ein Objekt zu zeichnen. Ist ein Objekt-Handler für ein Objekt vorhanden, wird der Aufruf der Funktion *OleDraw* vom Objekt-Handler empfangen und bearbeitet. Existiert kein Objekt-Handler, benutzt die Client-Bibliothek die Präsentationsdaten des Objekts, um es anzuzeigen oder zu drucken.

Sind die Präsentationsdaten des Objekts ein Metafile, übergibt die Bibliothek der Callback-Funktion der Client-Anwendung in gleichbleibenden Abständen die Benachrichtigung *OLE_QUERY_PAINT*, während das Objekt gezeichnet wird. Liefert die Callback-Funktion den Wert *FALSE*, kehrt die Funktion *OleDraw* unmittelbar zurück und das Zeichnen wird beendet. Ein Client könnte auch die Benachrichtigung *OLE_QUERY_PAINT* verwenden, um bestimmte Aktionen innerhalb der Callback-Funktion auszuführen und dann den Wert *TRUE* zurückliefern, damit das Zeichnen fortgesetzt wird. Die Aktionen, die während dieses Zeitraums von der Client-Anwendung ausgeführt werden, sollten das Zeichnen nicht beeinflussen (z.B. sollte kein Bildlauf im Fenster durchgeführt werden).

Wird das Zielgerät für ein Objekt gewechselt (zum Beispiel, wenn der Benutzer einen anderen Drucker auswählt), sollte die Client-Anwendung die Funktion *OleSetTargetDevice* aufrufen. Die Client-Anwendung sollte die Funktion *OleSetTargetDevice* auch dann aufrufen, wenn ein Objekt erzeugt oder geladen wird.

Ändert sich die Größe des Präsentationsrechtecks für das Objekt (z.B. durch eine Aktion des Benutzers), sollte die Client-Anwendung die Funktion *OleSetBounds* aufrufen. Nach dem Aufruf von *OleSetBounds* sollte die Client-Anwendung die Funktion *OleUpdate* aufrufen, um das Objekt zu aktualisieren und die Funktion *OleDraw*, um das Objekt neu zu zeichnen.

Objekte öffnen und schließen

Fordert der Benutzer die Client-Anwendung zur Aktivierung eines Objekts auf, sollte die Client-Anwendung durch einen Aufruf der Funktion *OleQuery-*

ReleaseStatus ermitteln, ob das Objekt bereits beschäftigt ist. In diesem Fall sollte die Client-Anwendung entweder das Objekt nicht öffnen oder in eine Botschaftsschleife eintreten und auf die Benachrichtigung *OLE_RELEASE* warten.

Ist das Objekt nicht beschäftigt, sollte die Client-Anwendung die Funktion *OleActivate* aufrufen. Die Bibliothek benachrichtigt die Client-Anwendung sobald die Server-Anwendung geöffnet ist oder wenn ein Fehler auftritt.

Die Funktion *OleActivate* erlaubt es der Client-Anwendung zu bestimmen, ob das aktivierte Objekt in einem Fenster der Server-Anwendung angezeigt werden soll. Eine Client-Anwendung könnte das Fenster der Server-Anwendung verbergen, wenn ein Objekt automatisch aktualisiert wird.

Eine Client-Anwendung kann die Funktion *OleQueryOpen* verwenden, um zu ermitteln, ob ein angegebenes Objekt geöffnet ist. Mit der Funktion *OleClose* kann die Client-Anwendung ein geöffnetes Objekt wieder schließen. Beim Schließen eines Objekts wird die Verbindung zur Server-Anwendung beendet. Um eine so beendete Verbindung zwischen einem verknüpften Objekt und einer geöffneten Server-Anwendung wiederherzustellen, kann die Client-Anwendung die Funktion *OleReconnect* benutzen. Mit Hilfe der Funktion *OleRelease* kann die Client-Anwendung ein geöffnetes Objekt schließen und aus dem Speicher freigeben.

Aktiviert eine Client-Anwendung zum erstenmal ein bestimmtes eingebettetes Objekt, sollte sie die Funktion *OleSetHostNames* aufrufen, um den String zu bestimmen, der in der Titelleiste des Fensters der Server-Anwendung angezeigt werden soll. Dieser String sollte der Name des Client-Dokuments sein, in dem sich das Objekt befindet. Die Client-Anwendung muß nicht jedesmal die Funktion *OleSetHostNames* aufrufen, wenn ein eingebettetes Objekt aktiviert wird, da die Bibliothek die angegebenen Namen aufzeichnet.

Objekte löschen

Um ein Objekt endgültig aus einem Dokument zu entfernen, sollte die Client-Anwendung die Funktion *OleDelete* aufrufen. Diese Funktion schließt das angegebene Objekt, falls nötig, bevor es entfernt wird.

Die Client-Befehle Ausschneiden und Kopieren

Eine Client-Anwendung kann ein Objekt in die Zwischenablage kopieren, indem sie die Zwischenablage öffnet, die Funktion *OleCopyToClipboard* aufruft und die Zwischenablage wieder schließt.

Unterstützt die Client-Anwendung eine verzögerte Übergabe, sollte sie folgende Schritte ausführen, um ein Objekt in die Zwischenablage zu kopieren oder auszuschneiden:

1. Die Zwischenablage öffnen und leeren.
2. Die gewünschten Datenformate in die Zwischenablage legen.

3. Die Funktion *OleEnumFormats* aufrufen, um die Formate für das Objekt abzurufen.

4. Die Funktion *SetClipboardData* aufrufen, um die Formate in der Zwischenablage abzulegen. Der Wert *NULL* bestimmt dabei das Handle der Daten. Ruft die Funktion *OleEnumFormats* das Format ObjectLink ab, sollte die Client-Anwendung die Funktion *SetClipboardData* mit OwnerLink anstelle des Formats ObjectLink aufrufen. Weitere Informationen finden Sie in der Beschreibung der Funktion *OleCopyToClipboard*.

5. Zusätzliche Präsentationsdaten in der Zwischenablage ablegen.

6. Die Zwischenablage schließen.

Um den Befehl *Ausschneiden* im Menü *Bearbeiten* zu unterstützen, kann eine Anwendung die Funktion *OleCopyToClipboard* aufrufen und dann das Objekt mit Hilfe der Funktion *OleDelete* entfernen. Die Client-Anwendung kann immer nur eines der ausgewählten Objekte in die Zwischenablage ablegen, auch wenn der Benutzer mehrere Objekte ausgewählt und ausgeschnitten oder kopiert hat. In diesem Fall legt die Client-Anwendung normalerweise das erste ausgewählte Objekt in die Zwischenablage.

Die Funktion *OleCopyToClipboard* kopiert immer OwnerLink-Formate und keine ObjectLink-Formate in die Zwischenablage. Bei eingebetteten Objekten stehen die Native-Daten immer vor den OwnerLink-Formaten. Falls ein verknüpftes Objekt Native-Daten verwendet, steht das OwnerLink-Format immer vor den Native-Daten. Benutzt eine Anwendung die Funktion *OleGetData*, um Daten von einem verknüpften Objekt abzurufen, das mit der Funktion *OleCopyToClipboard* in die Zwischenablage kopiert wurde, sollte sie das Format ObjectLink und nicht OwnerLink angeben, auch wenn das Format OwnerLink in die Zwischenablage gelegt wurde.

Kopiert eine Anwendung, die sowohl als Server- als auch als Client-Anwendung eingesetzt werden kann, eine Auswahl in die Zwischenablage, die ein oder mehrere Objekte enthält, sollte sie zuerst genügend Speicherplatz für die Auswahl reservieren. Den benötigten Speicherplatz für jedes Objekt kann die Anwendung mit der Funktion *OleQuerySize* ermitteln.

Wurde der Speicherplatz reserviert, sollte die Anwendung die Funktion *OleRegisterClientDoc* aufrufen und dabei »Clipboard« für den Namen des Dokuments angeben. In diesem Fall bezeichnet das durch den Aufruf der Funktion *OleRegisterClientDoc* zurückgelieferte Handle ein Dokument, das nur während des Kopiervorgangs benutzt wird. Die Anwendung ruft dann für jedes Objekt die Funktion *OleClone* auf, um eine Kopie des Objekts zu erhalten, die nicht mehr mit dem Server verknüpft ist, und anschließend die Funktion *OleSaveToStream*, um die so erhaltene Kopie in den reservierten Speicher zu schreiben. Danach kann mit der Funktion *OleRelease* der von der Kopie belegte Speicher wieder freigegeben werden. Ist die gesamte Auswahl in den Stream gesichert, kann die Anwendung die Funktion *SetClipboardData* aufrufen. Wurde diese Funktion erfolgreich ausgeführt, sollte die Anwendung die Funktion *OleSavedClientDoc* aufrufen. Danach ruft die Anwendung die Funktion *OleRevokeClientDoc* auf und gibt das Handle an, das von der Funktion *OleRegisterClientDoc* zurückgeliefert wurde.

Weitere Informationen zu den Befehlen *Ausschneiden* und *Kopieren* finden Sie im Abschnitt »Die Server-Befehle *Ausschneiden* und *Kopieren*«.

Objekte erzeugen

Eine Client-Anwendung kann verknüpfte und eingebettete Objekte in ein Dokument aufnehmen, indem sie die Objekte aus der Zwischenablage einfügt, sie aus einer Datei erzeugt, von anderen Objekten kopiert oder eine Server-Anwendung startet, um die Objekte direkt zu erzeugen.

Funktionen zum Erzeugen von Objekten

Jede der folgenden Funktionen erzeugt in einem angegebenen Dokument ein eingebettetes oder verknüpftes Objekt:

Funktion	Beschreibung
OleClone	Erzeugt eine exakte Kopie eines Objekts.
OleCopyFromLink	Erzeugt ein eingebettetes Objekt, indem sie ein verknüpftes Objekt kopiert.
OleCreate	Erzeugt ein eingebettetes Objekt der angegebenen Klasse.
OleCreateFromClip	Erzeugt ein Objekt aus der Zwischenablage. Diese Funktion erzeugt normalerweise ein eingebettetes Objekt.
OleCreateFromFile	Erzeugt ein Objekt aus dem Inhalt einer Datei. Diese Funktion erzeugt normalerweise ein eingebettetes Objekt.
OleCreateFromTemplate	Erzeugt ein eingebettetes Objekt, indem ein anderes Objekt als Schablone benutzt wird.
OleCreateInvisible	Erzeugt ein Objekt, ohne die Server-Anwendung für den Benutzer anzuzeigen.
OleCreateLinkFromClip	Erzeugt ein Objekt, indem Informationen aus der Zwischenablage verwendet werden. Diese Funktion erzeugt normalerweise ein verknüpftes Objekt.
OleCreateLinkFromFile	Erzeugt ein Objekt aus dem Inhalt einer Datei. Diese Funktion erzeugt normalerweise ein verknüpftes Objekt.
OleObjectConvert	Erzeugt ein Objekt, das ein angegebenes Protokoll unterstützt, indem ein bereits vorhandenes Objekt konvertiert wird.

Jede dieser Funktionen benötigt einen Parameter, der bei ihrer Rückkehr auf eine *OLEOBJECT*-Struktur zeigt. Server-Anwendungen erstellen bei der Erzeugung eines Objekts meistens eine *OLEOBJECT*-Struktur. *OLEOBJECT* zeigt auf Funktionen, die die Interaktion von Server-Anwendung und Objekt beschreiben. Bevor die Client-Bibliothek der Client-Anwendung einen Zeiger auf diese

Struktur übergibt, fügt die Bibliothek der Struktur einige interne Informationen entsprechend der OwnerLink- oder ObjectLink-Daten hinzu. Diese internen Informationen ermöglichen es der Client-Bibliothek, die richtige Server-Anwendung zu bestimmen, wenn ihr von einer OLE-Funktion wie zum Beispiel *OleActivate* ein Zeiger auf eine *OLEOBJECT*-Struktur übergeben wird. Weitere Informationen über die *OLEOBJECT*-Struktur erhalten Sie im Abschnitt »Eine Server-Anwendung starten«.

Jedes neue Objekt muß einen Namen besitzen, der für das Client-Dokument eindeutig ist. Obwohl beschreibende Objektnamen hilfreich sein können, weisen einige Anwendungen neuen Objektnamen einfach eine Zahl zu, die bei jedem neuen Objekt um eins erhöht wird. Weitere Informationen über Objektnamen finden Sie im Abschnitt »Dokumente verwalten«.

Implementiert eine Client-Anwendung den Befehl *Objekt einfügen*, sollte sie die Registrierungsdatenbank verwenden, um die verfügbaren OLE-Server zu ermitteln und diese dann dem Benutzer auflisten. Wählt der Benutzer eine Server-Anwendung aus und betätigt den *OK*-Schalter, kann die Client-Anwendung die Funktion *OleCreate* verwenden, um ein Objekt an der aktuellen Position zu erzeugen.

Die Funktionen *OleCopyFromLink, OleCreate* und *OleCreateFromTemplate* erzeugen immer ein eingebettetes Objekt. Die anderen Funktionen zum Erzeugen von Objekten können entweder ein eingebettetes oder ein verknüpftes Objekt erzeugen, je nach Reihenfolge und Art der verfügbaren Daten.

Empfängt die Callback-Funktion der Client-Anwendung die Benachrichtigung *OLE_RELEASE*, nachdem die Client-Anwendung die Funktion *OleCreate* oder *OleCreateFromFile* aufgerufen hat, sollte die Client-Anwendung diese Botschaft mit dem Aufruf der Funktion *OleQueryReleaseError* beantworten. Zeigt die Funktion *OleQueryReleaseError* an, daß bei der Erzeugung des Objekts ein Fehler aufgetreten ist, sollte die Client-Anwendung das Objekt entfernen.

Wenn eine Funktion zum Erzeugen von Objekten *OLE_WAIT_FOR_RELEASE* zurückgibt, sollte die aufrufende Anwendung entweder auf die Benachrichtigung *OLE_RELEASE* warten oder den Benutzer benachrichtigen, daß das Objekt nicht erzeugt werden kann. Weitere Informationen erhalten Sie im Abschnitt »Asynchrone Operationen«.

Nimmt eine Client-Anwendung Dateien entgegen, die vom Datei-Manager abgelegt werden, sollte sie die Botschaft *WM_DROPFILES* durch einen Aufruf der Funktion *OleCreateFromFile* beantworten und »Packager« für den Parameter *lpszClass* angeben.

Die Befehle Einfügen und Verknüpfung einfügen

Eine Client-Anwendung sollte folgende Schritte durchführen, um ein eingebettetes oder verknüpftes Objekt durch Einfügen aus der Zwischenablage zu erzeugen.

1. Aufruf der Funktion *OleQueryCreateFromClip*, um zu bestimmen, ob der Befehl *Einfügen* aktiviert werden soll. Schlägt die Funktion bei Angabe von StdFileEditing für den Parameter *lpszProtocol* fehl, sollte beim erneuten Aufruf Static angegeben werden.

2. Aufruf der Funktion *OleQueryLinkFromClip*, um zu bestimmen, ob der Befehl *Verknüpfung einfügen* aktiviert werden soll.

- Der Benutzer wählt den Befehl *Einfügen*: Öffnen der Zwischenablage und Aufruf der Funktion *OleCreateFromClip*.
- Der Benutzer wählt den Befehl *Verknüpfung einfügen*: Öffnen der Zwischenablage und Aufruf der Funktion *OleCreateLinkFromClip*.

3. Die Zwischenablage schließen.

4. Aufruf der Funktion *OleQueryType*, um zu bestimmen, welche Objektart durch die Funktion erzeugt wird. In Abhängigkeit von der Reihenfolge der Daten in der Zwischenablage kann die Funktion *OleCreateFromClip* auch ein verknüpftes Objekt und die Funktion *OleCreateLinkFromClip* auch ein eingebettetes Objekt erzeugen.

Die Client-Anwendung sollte die eingefügten Daten oder das Objekt an der aktuellen Position im Dokument speichern. Sie sollte das Objekt auswählen (markieren), damit der Benutzer es sofort bearbeiten kann. Schlägt sowohl die Funktion *OleQueryCreateFromClip* als auch die Funktion *OleQueryLinkFromClip* fehl und befinden sich Daten in der Zwischenablage, die die Client-Anwendung interpretieren kann, sollte letztere den Befehl *Einfügen* aktivieren.

Sind die Informationen in der Zwischenablage nicht vollständig – wenn zum Beispiel Native-Daten ohne das OwnerLink-Format angegeben sind – sollte der Befehl *Einfügen* ein statisches Objekt im Dokument einfügen. Ein statisches Objekt besteht aus Präsentationsdaten für ein Objekt; es kann mit Hilfe der normalen OLE-Techniken nicht bearbeitet werden. Der Versuch der Öffnung eines statischen Objekts schlägt fehl, und es wird keine Benachrichtigung erzeugt.

Implementiert die Client-Anwendung den Befehl *Einfügen mit Formatauswahl*, sollte sie die Funktion *EnumClipboardFormats* verwenden, um eine Liste der Datenformate in der Zwischenablage zu erzeugen. Die Client-Anwendung sollte auch die Registrierungsdatenbank überprüfen, um den vollständigen Namen der Server-Anwendung zu ermitteln. Der Schalter *Verknüpfung einfügen* im Dialogfenster *Einfügen mit Formatauswahl* entspricht dem Befehl *Verknüpfung einfügen* im Menü *Bearbeiten*.

Ist das DDE-Format Link anstelle des ObjectLink-Formats in der Zwischenablage verfügbar, sollte die Client-Anwendung dieselbe Verknüpfungsoperation durchführen, die sie vor der Implementierung von OLE unterstützt hat.

Der Befehl Rückgängig

Eine Client-Anwendung kann die Funktion *OleClone* verwenden, um den Befehl *Rückgängig* zu unterstützen. Ein mit *OleClone* kopiertes Objekt ist bis auf die bestehenden Verbindungen zur Server-Anwendung identisch mit dem Original. Ein so kopiertes Objekt ist nicht automatisch mit der Server-Anwendung verbunden.

Wird die Server-Anwendung geschlossen und das Objekt aktualisiert, kann der Benutzer mit Hilfe der gesicherten Kopie des Objekts alle in der Server-Anwendung durchgeführten Änderungen am Objekt rückgängig machen. Die

Unterstützung des Befehls *Rückgängig* wird von der Client-Anwendung bereitgestellt, da die Server-Anwendung den früheren Zustand von Objekten nicht verwalten kann.

Der Befehl *Rückgängig* stellt den Zustand des Objekts vor der letzten Aktualisierung durch die Server-Anwendung wieder her. Um dies zu ermöglichen, muß die Client-Anwendung das Objekt mit *OleClone* kopieren, wenn es erstmals aktiviert wird und dann das aktualisierte Objekt, wenn eine Aktualisierung stattfindet.

Die Client-Anwendung muß zwei Kopien des Objekts verwalten. Die Kopie des Originals muß erhalten bleiben, damit ein aktualisiertes Objekt wiederhergestellt werden kann, wenn der Benutzer den Befehl *Rückgängig* auswählt. Die Kopie des aktualisierten Objekts wird benötigt, um den Befehl *Rückgängig* ausführen zu können, wenn ein aktualisiertes Objekt erneut aktualisiert wird. Da sich die Daten eines Objekts bei einer Aktualisierung ändern, muß die Kopie des Objekts für den Befehl *Rückgängig* noch vor der Aktualisierung angefertigt werden.

Da die Client-Anwendung nicht zwischen den verschiedenen Formen der Aktivierung eines Objekts unterscheiden kann, muß sie auch für die Aktionsarten, die das Objekt nicht bearbeiten, eine Kopie des Objekts anlegen, selbst wenn in diesen Fällen keine Aktualisierung des Objekts erfolgen kann.

Der Befehl <Klassenname> Objekt

Eine Client-Anwendung kann den Befehl *<Klassenname> Objekt* implementieren, indem sie die Funktion *OleActivate* verwendet. Die Funktion *OleActivate* besitzt einen Parameter, der es der Client-Anwendung ermöglicht, die vom Benutzer ausgewählte Aktionsart anzugeben.

Der Befehl Verknüpfungen

Wählt ein Benutzer den Befehl *Verknüpfungen*, wird ein Dialogfenster geöffnet, in dem jedes verknüpfte Objekt innerhalb des Dokuments aufgeführt wird. Die gewählten Verknüpfungen werden im Dialogfenster hervorgehoben dargestellt. Das Dialogfenster ermöglicht es dem Benutzer, die Aktionsarten zu einem Objekt aufzurufen, zu entscheiden, ob die Aktualisierung einer Verknüpfung automatisch oder manuell erfolgen soll, ob eine Verknüpfung sofort aktualisiert werden soll, ob eine Verknüpfung abgebrochen und ob unterbrochene Verknüpfungen wiederhergestellt werden sollen.

Das Dialogfenster *Verknüpfungen* enthält Schalter, die es dem Benutzer ermöglichen, die primären und sekundären Aktionsarten für ein Objekt zu aktivieren. Eine Client-Anwendung kann diese Schalter implementieren, indem sie die Funktion *OleActivate* verwendet.

Eine Client-Anwendung kann die Funktionen *OleGetLinkUpdateOptions* und *OleSetLinkUpdateOptions* verwenden, um die Schaltfelder zur Aktualisierung von

Verknüpfungen im Dialogfenster zu unterstützen. Nachstehend finden Sie die drei möglichen Aktualisierungsoptionen:

Option	Beschreibung
oleupdate_always	Aktualisiert das verknüpfte Objekt, sobald es möglich ist. Diese Option unterstützt das Schaltfeld *Automatische Aktualisierung* im Dialogfenster.
oleupdate_onsave	Aktualisiert das verknüpfte Objekt, sobald das Quelldokument von der Server-Anwendung gesichert wird.
oleupdate_oncall	Aktualisiert das verknüpfte Objekt nur auf Anforderung der Client-Anwendung. Diese Option unterstützt das Schaltfeld *Manuelle Aktualisierung* im Dialogfenster *Verknüpfungen*.

Diese Aktualisierungsoptionen steuern, wann eine Aktualisierung der Präsentation eines Objekts erfolgt. Die Inhalte des Quelldokuments werden verwendet, um die Präsentation zu aktualisieren, sobald die Verknüpfung aktiviert wird.

Um den Schalter *Jetzt aktualisieren* im Dialogfenster *Verknüpfungen* zu unterstützen, kann eine Anwendung die Funktion *OleUpdate* aufrufen. Wählt ein Benutzer den Befehl *Jetzt aktualisieren*, sollte die Client-Anwendung die vom Benutzer ausgewählten Verknüpfungen aktualisieren.

Die Auswahl des Schalters *Verknüpfung abbrechen* im Dialogfenster *Verknüpfungen* macht aus einem Objekt ein Bild, das eine Anwendung mit OLE-Standardtechniken nicht bearbeiten kann. Eine Anwendung kann diesen Schalter mit der Funktion *OleObjectConvert* implementieren.

Eine Client-Anwendung sollte den Schalter *Verknüpfung abbrechen* im Dialogfenster *Verknüpfungen* nur dann aktivieren, wenn sich alle ausgewählten Verknüpfungen auf dasselbe Quelldokument beziehen. Besitzt die Client-Anwendung die nötige Information, kann sie die Verknüpfung mit den Funktionen *OleGetData* und *OleSetData* wiederherstellen. Um die Verknüpfungsinformation für ein Objekt abzurufen, kann eine Client-Anwendung die Funktion *OleGetData* aufrufen und beim Aufruf das ObjectLink-Format angeben. Der Aufruf der Funktion *OleGetData* schlägt fehl, falls das ObjectLink-Format angegeben wird und das Objekt keine Verknüpfung ist. Eine Client-Anwendung kann Klasseninformationen abrufen, indem sie die Funktion *OleGetData* verwendet und dabei entweder das OwnerLink-Format (für eingebettete Objekte) oder das ObjectLink-Format (für verknüpfte Objekte) angibt. Die Client-Anwendung kann es dem Benutzer ermöglichen, die Verknüpfungsinformation zu bearbeiten und im Objekt zu speichern, indem sie die Funktion *OleSetData* verwendet und dabei das ObjectLink-Format angibt.

Eine Client-Anwendung schließen

Eine Client-Anwendung sollte mit der Funktion *OleRelease* alle Objekte aus dem Speicher zu entfernen, wenn sie beendet wird. Wenn die Bibliothek statt

OLE_OK den Wert *OLE_WAIT_FOR_RELEASE* zurückgibt, sollte die Client-Anwendung nicht beendet werden. Bis zum Empfang der Benachrichtigung *OLE_RELEASE* kann die Client-Anwendung zum Beispiel Dateien schließen, Speicherplatz freigeben und Fenster verbergen.

Durch die Benachrichtigung *OLE_RELEASE* wird der Callback-Funktion der Client-Anwendung mitgeteilt, daß eine Operation in einer Server-Anwendung beendet wurde. Die Benachrichtigung bezeichnet jedoch nicht die Art der Operation oder ob diese erfolgreich ausgeführt wurde. Um dies zu bestimmen, kann eine Client-Anwendung die Funktion *OleQueryReleaseStatus* aufrufen. Die Funktion *OleQueryReleaseMethod* gibt die Art der an einem Objekt ausgeführten Operation an. Mit der Funktion *OleQueryReleaseError* kann die Client-Anwendung den Fehlerwert für die Operation ermitteln.

Ist die Zwischenablage der Client-Anwendung zugeordnet, wenn letztere beendet wird, sollte sie sicherstellen, daß die Daten in der Zwischenablage vollständig und in der richtigen Reihenfolge vorliegen.

Server-Anwendungen

Eine OLE-Server-Anwendung unterstützt Funktionen, die von der Server-Bibliothek aufgerufen werden, wenn ein Benutzer mit einem Objekt arbeitet. Die Server-Bibliothek OLESVR.DLL benutzt DDE-Befehle, um mit der Client-Bibliothek zu kommunizieren. Ruft die Client-Anwendung eine der Funktionen der OLE-API auf, benachrichtigt die Client-Bibliothek zunächst die Server-Bibliothek, die dann die Anforderung an die entsprechende Funktion in der vom Server bereitgestellten Liste mit Funktionszeigern weiterleitet.

Zusätzlich zu den vom Server erzeugten Funktionen, die von der Server-Bibliothek aufgerufen werden, gibt es zehn OLE-Funktionen zur Steuerung der Zugriffsmöglichkeiten der Bibliothek auf den Server und die von ihm verwalteten Dokumente und Objekte:

Funktion	Beschreibung
OleBlockServer	Stellt Anforderungen an die Server-Anwendung in eine Warteschlange, bis der Server die Funktion *OleUnblockServer* aufruft.
OleRegisterServer	Registriert die angegebene Server-Anwendung für die Bibliothek. Diese Informationen schließen den Klassennamen und die Instanz der Klasse ein und beschreiben, ob die Server-Anwendung Einfach- oder Mehrfachinstanzen unterstützt.
OleRegisterServerDoc	Registriert ein Dokument für die Server-Bibliothek.
OleRenameServerDoc	Gibt dem angegebenen Dokument einen neuen Namen.
OleRevertServerDoc	Stellt das Dokument in einem zuvor gesicherten Zustand wieder her, ohne das Dokument zu schließen.

Funktion	Beschreibung
OleRevokeObject	Hebt den Zugriff auf das angegebene Dokument auf.
OleRevokeServer	Hebt den Zugriff auf den angegebenen Server auf, schließt alle Dokumente und beendet die Kommunikationen mit den Client-Anwendungen.
OleRevokeServerDoc	Hebt den Zugriff auf das angegebene Dokument auf.
OleSavedServerDoc	Teilt der Bibliothek mit, daß ein Dokument gesichert wurde. Dieser Funktionsaufruf entspricht der Übergabe der Benachrichtigung *OLE_SAVED*.
OleUnblockServer	Bearbeitet eine Anforderung aus einer Warteschlange, die erzeugt wird, wenn die Server-Anwendung die Funktion *OleBlockServer* aufruft.

Die Funktionen *OleRevokeServer* und *OleRevokeServerDoc* können den Wert *OLE_WAIT_FOR_RELEASE* zurückliefern. Empfängt eine Server-Anwendung diesen Fehlerwert, sollte sie wie eine Client-Anwendung mit der Botschaftsbearbeitung weitermachen, bis die Server-Bibliothek die entsprechende *Release*-Funktion aufruft.

Eine Server-Anwendung starten

Der Start einer Server-Anwendung sollte folgendermaßen ausgeführt werden:

1. Registrieren der Fensterklassen und Fensterfunktionen für das Hauptfenster, die Dokumente und die Objekte.

2. Initialisieren der Funktionstabellen für die *OLESERVERVTBL*-, *OLESERVERDOCVTBL*- und *OLEOBJECTVTBL*-Strukturen.

3. Registrieren der Formate der Zwischenablage.

4. Reservieren von Speicherplatz für die Struktur *OLESERVER*.

5. Registrieren der Server-Anwendung für die Bibliothek durch Aufruf der Funktion *OleRegisterServer*.

6. Auf die Optionen */Embedding* (bzw. *-Embedding*) und */Embedding Dateiname* in der Kommandozeile prüfen und entsprechend der folgenden Richtlinien reagieren.

- Falls weder */Embedding* noch */Embedding Dateiname* vorhanden ist, muß die Funktion *OleRegisterServerDoc* aufgerufen und ein unbenanntes Dokument angegeben werden.

- Falls die Option */Embedding* vorhanden ist, darf weder ein Dokument registriert noch ein Fenster auf dem Bildschirm dargestellt werden. In diesem Fall arbeitet die Server-Anwendung nur nach Aufrufen durch die Server-Bibliothek.

- Falls die Option */Embedding Dateiname* vorhanden ist, darf kein Fenster auf dem Bildschirm dargestellt werden. Bearbeitung des String mit dem Dateinamen und Aufruf der Funktion *OleRegisterServerDoc*.

Die *OLESERVERVTBL*-, OLESERVERDOCVTBL- und *OLEOBJECTVTBL*-Strukturen sind Tabellen mit Zeigern auf Funktionen. Die Server-Bibliothek benutzt diese Strukturen, um Anforderungen von der Client-Anwendung zur Server-Anwendung weiterzuleiten. Die Server-Anwendung sollte die in diesen Strukturen enthaltenen Zeiger auf Funktionen mit der Funktion *MakeProcInstance* erzeugen. Die Funktionen sollten außerdem in die Moduldefinitionsdatei der Anwendung exportiert werden.

Die *OLESERVER*-Struktur enthält einen Zeiger auf eine *OLESERVERVTBL*-Struktur. Diese enthält Zeiger auf Funktionen zur Steuerung der grundlegenden Aufgaben eines Server wie das Öffnen von Dateien, Erzeugen von Objekten und Beenden der Server-Anwendung nach einer Bearbeitung. Einige der Funktionen, auf die ein Zeiger der *OLESERVERVTBL*-Struktur zeigt, veranlassen die Server-Anwendung zur Reservierung und Initialisierung einer *OLESERVERDOC*-Struktur.

Die *OLESERVERDOC*-Struktur enthält einen Zeiger auf eine *OLESERVERDOCVTBL*-Struktur. Diese Struktur enthält Zeiger auf Funktionen, die zum Beispiel das Speichern und Schließen von Dokumenten und das Setzen der Größe von Dokumenten steuern. Die *OLESERVERDOCVTBL*-Struktur enthält auch eine Funktion, die die Server-Anwendung zur Reservierung und Initialisierung einer *OLEOBJECT*-Struktur veranlaßt.

Die *OLEOBJECT*-Struktur enthält einen Zeiger auf eine *OLEOBJECTVTBL*-Struktur. Letztere enthält Zeiger auf Funktionen, die Operationen an Objekten ausführen. Nachdem die Server-Anwendung eine *OLEOBJECT*-Struktur erzeugt hat, gibt die Server-Bibliothek Informationen über die Struktur an die Client-Bibliothek weiter. Daraufhin erzeugt die Client-Bibliothek eine parallele *OLEOBJECT*-Struktur (mit internen Informationen, die die Server-Anwendung, das Dokument und das Element für das Objekt bezeichnen) und übergibt einen Zeiger auf diese Struktur an die Client-Anwendung.

Durch diese Hierarchie der Strukturen *OLESERVER*, *OLESERVERDOC* und *OLEOBJECT* kann eine Server-Anwendung soviele Dokumente öffnen, wie die Bibliothek anfordert. Außerdem kann jedes Dokument jede benötigte Anzahl Objekte enthalten.

Durch Aufruf der Funktion *RegisterClipboardFormat* kann eine Server-Anwendung die Formate der Zwischenablage (Native-, OwnerLink- und ObjectLink-Format oder auch alle anderen erforderlichen Formate) registrieren.

Startet eine Server-Anwendung, erzeugt sie eine *OLESERVER*-Struktur und registriert diese durch einen Aufruf der Funktion *OleRegisterServer* für die Bibliothek. Sobald diese Funktion zurückkehrt, zeigt einer ihrer Parameter auf ein Server-Handle. Die Bibliothek benutzt dieses Handle als Referenz auf die Server-Anwendung und die Server-Anwendung benutzt dieses Handle in Aufrufen server-spezifischer OLE-Funktionen.

Ist eine OLE-Server-Anwendung zugleich eine DDE-Server-Anwendung, darf der Klassenname, der beim Aufruf der Funktion *OleRegisterServer* angegeben wird, nicht identisch mit dem Namen der ausführbaren Datei der Anwendung sein.

Wenn eine Client-Anwendung, die mit einem Verbunddokument arbeitet, ein verknüpftes oder eingebettetes Objekt zum Bearbeiten öffnet, startet die Client-Bibliothek den Server mit der Kommandozeilenoption */Embedding*. Mit dieser Option kann die Server-Anwendung bestimmen, ob das Objekt direkt durch einen Benutzer oder im Rahmen von OLE geöffnet wurde. Bei einem verknüpften Objekt muß der Option */Embedding* der Dateiname folgen. Wird eine Server-Anwendung für ein eingebettetes Objekt mit der Option */Embedding* gestartet, sollte sie weder ein Dokument erzeugen noch ein Fenster anzeigen. Stattdessen sollte die Server-Anwendung die Funktion *OleRegisterServer* aufrufen und dann in eine Schleife zur Bearbeitung von Botschaften eintreten. Beim Starten der Server-Anwendung mit der Option */Embedding Dateiname* sollte die Anwendung außerdem die Funktion *OleRegisterServer* aufrufen. Die Server-Anwendung arbeitet dann entsprechend der Aufrufe der Bibliothek. Die Server-Anwendung sollte solange nicht sichtbar werden, bis die Bibliothek die Funktion *Show* oder *DoVerb* der *OLEOBJECTVTBL*-Struktur aufruft. Server-Anwendungen sollten sowohl auf die Option *-Embedding* als auch auf */Embedding* prüfen.

Durch Aufruf der Funktion *OleBlockServer* kann eine Server-Anwendung Anforderungen der Client-Bibliothek in eine Warteschlange stellen. Ist die Server-Anwendung zur Bearbeitung der Anforderungen der Server-Bibliothek bereit, kann sie die Funktion *OleUnblockServer* aufrufen. Die Funktion *OleUnblockServer* sollte vor der Funktion *GetMessage* in einer Botschaftsschleife aufgerufen werden, damit alle zurückgestellten Anforderungen freigegeben werden, bevor die nächste Botschaft eingeht. Oftmals gibt eine Server-Anwendung *OLE_BUSY* zurück, anstatt die Funktion *OleBlockServer* aufzurufen. Dies hat zwei Vorteile: Zum einen kann die Client-Anwendung entscheiden, ob die Botschaft wiederholt oder die Operation eingestellt werden soll. Zum anderen kann die Server-Anwendung entscheiden, welche Anforderungen bearbeitet werden sollen.

Tritt in einer vom Server bereitgestellten Funktion ein Fehler auf, sollte die Server-Anwendung den *OLESTATUS*-Fehlerwert zurückgeben, der den Fehler am besten beschreibt. Die OLE-Bibliotheken verwenden diese Fehlerwerte, um eine geeignete Reaktion in Fehlersituationen zu bestimmen. Es ist jedoch möglich, daß die Client-Anwendung die vom Server zurückgelieferten Fehlerwerte nicht empfängt; die OLE-Bibliotheken können Fehlerwerte ändern, bevor sie an die Client-Anwendung übergeben werden.

Ein Dokument oder ein Objekt öffnen

Sobald die Server-Bibliothek die Funktion *Open*, *Create*, *CreateFromTemplate* oder *Edit* in der *OLESERVERVTBL*-Struktur aufruft, erzeugt die Server-Anwendung eine *OLESERVERDOC*-Struktur. Wird das Dokument durch einen Aufruf der Server-Bibliothek geöffnet, übergibt die Server-Anwendung die *OLESERVERDOC*-Struktur an die Bibliothek. Falls das Dokument jedoch direkt von einem Benutzer geöffnet wird, sollte die Server-Anwendung die Funktion *OleRegisterServerDoc* aufrufen, um das Dokument für die Bibliothek zu registrieren. Mit

Hilfe der Funktion *GetObject* in der *OLESERVERDOCVTBL*-Struktur fordert die Bibliothek die Server-Anwendung auf, für jedes von der Client-Anwendung angeforderte Objekt eine *OLEOBJECT*-Struktur zu erzeugen.

Normalerweise wird eine neue Instanz der Server-Anwendung gestartet, wenn die Client-Anwendung ein verknüpftes oder eingebettetes Objekt aktiviert. Diese neue Instanz ist nicht nötig, wenn das Objekt bereits in einer anderen Instanz der Server-Anwendung geöffnet ist oder wenn die Server-Anwendung eine bereits geöffnete MDI-Server-Anwendung ist.

Ob die Server-Bibliothek eine neue Instanz der Server-Anwendung startet, um ein eingebettetes oder verknüpftes Objekt zu bearbeiten, hängt von dem Wert ab, der beim Aufruf der Funktion *RegisterServer* angegeben wird.

Die Server-Befehle Ausschneiden und Kopieren

Mit folgenden Schritten kann eine Server-Anwendung Daten in die Zwischenablage kopieren oder ausschneiden, mit denen die Client-Anwendung ein eingebettetes oder verknüpftes Objekt erzeugen kann:

1. Öffnen und Leeren der Zwischenablage.

2. Ablegen der die Auswahl beschreibenden Datenformate in der Zwischenablage mit Hilfe der Funktion *SetClipboardData*.

3. Schließen der Zwischenablage.

Wenn die Server-Anwendung Daten in die Zwischenablage ausschneidet statt sie zu kopieren, werden normalerweise keine ObjectLink- oder Link-Formate angeboten, da die Quelle der Daten aus dem Dokument entfernt wurde.

Die Server-Anwendung sollte Daten in der Reihenfolge in der Zwischenablage ablegen, die im Abschnitt »Konventionen der Zwischenablage« angegeben ist.

Normalerweise legt die Server-Anwendung server-spezifische Formate, Native-Formate, OwnerLink-Formate und Präsentationsformate in der Zwischenablage ab.

Kann die Server-Anwendung Verknüpfungen unterstützen, legt sie auch ObjectLink-Formate und gegebenenfalls Link-Formate in der Zwischenablage ab. Die Server-Anwendung muß ein Präsentationsformat (*CF_METAFILE*, *CF_BITMAP* oder *CF_DIB*) zur Verfügung stellen, wenn sie keinen Objekt-Handler besitzt. Native-Daten können nur dann als Präsentationsformat verwendet werden, wenn die Server-Anwendung einen Objekt-Handler besitzt, der die Native-Daten verarbeiten kann.

Kopiert ein Benutzer eine Auswahl in die Zwischenablage, die ein eingebettetes Objekt oder eine Verknüpfung enthält, so hängen die von der Server-Anwendung zu kopierenden Datenformate davon ab, ob das Verbunddokument das Objekt oder die Verknüpfung ändert.

Wenn es zu keiner Veränderung des Objekts oder der Verknüpfung kommt, sind die Native- und OwnerLink-Formate aus der ursprünglichen Quelle des Objekts die besten Formate.

Andernfalls – zum Beispiel wenn andere Farben zugewiesen werden – sollten

die Native- und OwnerLink-Formate aus dem Verbunddokument verwendet werden.

Wenn eine Server-Anwendung ein Metafile als Präsentationsformat für ein Objekt benutzt, muß der Abbildungsmodus für dieses Metafile *MM_ANISOTROPIC* sein. Verwendet eine Server-Anwendung Schriften in diesem Metafile, kann mit TrueType-Schriften die Leistung erhöht werden. Metafiles können besser skaliert werden, wenn sie TrueType-Schriften benutzen. Um ausschließlich TrueType-Schriften zu verwenden, muß die Server-Anwendung das zweite Bit (0x04) des Elements *lpPitchAndFamily* der *LOGFONT*-Struktur setzen.

Die OLE-Bibliotheken geben die Größe jedes Objekts in *MM_HIMETRIC*-Einheiten an. Dabei sollte weder die Breite noch die Höhe eines Objekts 32767 *MM_HIMETRIC*-Einheiten überschreiten.

Die Befehle Aktualisieren, Speichern unter und Neu

Wird die Server-Anwendung im Rahmen der Bearbeitung eines Objekts innerhalb eines Verbunddokuments gestartet, sollte sie den Befehl *Speichern* im Menü *Datei* in den Befehl *Aktualisieren* umwandeln. Wenn der Benutzer den Befehl *Aktualisieren* auswählt, sollte die Server-Anwendung die Funktion *OleSavedServerDoc* aufrufen.

Wählt der Benutzer den Befehl *Speichern unter*, *Neu* oder *Öffnen* in einer Server-Anwendung für ein Einzeldokument, sollte die Anwendung zunächst rückfragen, ob das Objekt im Verbunddokument aktualisiert werden soll, bevor die Aktion ausgeführt wird. Bei Auswahl des Befehls *Speichern unter* sollte die Server-Anwendung die Funktion *OleRenameServerDoc* aufrufen. Entscheidet sich der Benutzer jedoch bei der Rückfrage der Server-Anwendung dafür, daß zuerst Änderungen am Objekt abgespeichert werden sollen, bevor das Dokument umbenannt wird, sollte die Server-Anwendung vor der Funktion *OleRenameServerDoc* die Funktion *OleSavedServerDoc* aufrufen. Bei eingebetteten Objekten wird durch Auswahl des Befehls *Speichern unter* die Verbindung zur Client-Anwendung gelöst, da mit diesem Befehl das Dokument im Speicher der angegebenen neuen Datei zugeordnet wird. Der Aufruf der Funktion *OleRenameServerDoc* für verknüpfte Objekte nach der Auswahl des Befehls *Speichern unter* ermöglicht es der Client-Anwendung, die Verknüpfung der neuen Datei zuzuordnen.

Die meisten Server-Anwendungen verwalten ein Änderungs-Flag, das gelegentlich auch »dirty« genannt wird. Dieses Flag zeigt an, ob in einem geöffneten Dokument einer Instanz Änderungen vorgenommen wurden. In der nachfolgenden Tabelle sind die für das Flag geltenden Regeln aufgelistet, wenn die Server-Anwendung ein eingebettetes Objekt bearbeitet. Anhand dieser Regeln kann eine Server-Anwendung sicherstellen, daß dieses Flag nur dann den Wert *TRUE* hat, wenn das von der Server-Anwendung bearbeitete Dokument dem eingebetteten Objekt in der Client-Anwendung entspricht. Andernfalls hat das Flag den Wert *FALSE*.

Flag	Bedingung
TRUE	Die Bibliothek ruft die Funktion *Create* der *OLESERVERVTBL*-Struktur auf.
TRUE	Die Bibliothek ruft die Funktion *CreateFromTemplate* der *OLESERVER-VTBL*-Struktur auf.
TRUE	Das Dokument wird in der Server-Anwendung geändert.
FALSE	Die Bibliothek ruft die Funktion *Edit* der *OLESERVERVTBL*-Struktur auf.
FALSE	Die Bibliothek ruft die Funktion *GetData* der *OLEOBJECTVTBL*-Struktur mit dem Native-Datenformat auf (für alle anderen Formate sollte sich das Flag nicht ändern).

Wenn das Änderungs-Flag den Wert *TRUE* hat und die Server-Anwendung ein Dokument freigibt, in dem ein eingebettetes Objekt bearbeitet wurde, sollte sie entsprechend dieser Regeln ein Meldungsfenster darstellen. Der Benutzer kann dann wählen, ob das Objekt aktualisiert werden soll.

In einer MDI-Server-Anwendung öffnen die Befehle *Neu* und *Öffnen* im Menü *Datei* ein neues Fenster. Die Verbindung mit der Client-Anwendung bleibt unverändert. Nachdem der Benutzer einen dieser Befehle angewählt hat, kann er mit der Server-Anwendung weiterarbeiten. Beendet er jedoch die Server-Anwendung, kehrt der Fokus nicht notwendigerweise zur Client-Anwendung zurück.

Wenn ein Objekt im Client-Dokument aktualisiert werden muß oder die Server-Anwendung das Dokument schließt, kann sie die Funktion *OleSaved-ServerDoc* aufrufen. Schließt die Server-Anwendung das Dokument, und das Objekt sollte aktualisiert werden, so sendet sie die Benachrichtigung *OLE_-CLOSED*. Client-Anwendungen erhalten diese Benachrichtigung nur für eingebettete und nicht für verknüpfte Objekte, da die Server-Bibliothek die Benachrichtigung für verknüpfte Objekte abfängt.

Eine Server-Anwendung schließen

Muß die Server-Anwendung beendet werden, ruft die Server-Bibliothek die Funktion *Exit* der *OLESERVERVTBL*-Struktur auf. Die Server-Bibliothek ruft die Funktion *Release* auf, um der Server-Anwendung mitzuteilen, daß die Ausführung beendet werden kann. Jedoch beendet die Server-Anwendung bei diesem Funktionsaufruf nicht unbedingt ihre Ausführung.

Wenn die Server-Anwendung im Hintergrund läuft, muß sie beim Aufruf von *Release* ihre Ausführung beenden. Die einzige Ausnahme besteht darin, wenn eine Anwendung mehrfach als Server-Anwendung dient. Dann kann auch eine im Hintergrund laufende Server-Anwendung nicht immer mit dem Aufruf von *Release* beendet werden. Hat die Server-Anwendung keine geöffneten Dokumente und wurde sie mit der Option */Embedding* (also von einem Client) gestartet, sollte sie beim Aufruf von *Release* beendet werden. Lädt der Benutzer jedoch ausdrücklich ein Dokument in eine Einfachinstanzen-(MDI)-Server-Anwen-

dung, sollte die Server-Anwendung beim Aufruf von *Release* durch die Bibliothek nicht beendet werden.

Wenn der Benutzer eine Server-Anwendung schließt, die ein eingebettetes Objekt bearbeitet hat, ohne vorher die Client-Anwendung mit den durchgeführten Änderungen zu aktualisieren, sollte die Server-Anwendung beim Benutzer rückfragen, ob die Änderungen gespeichert werden sollen. Bestätigt der Benutzer die Rückfrage, sollte die Server-Anwendung die Benachrichtigung *OLE_CLOSED* senden und die Funktion *OleRevokeServerDoc* aufrufen. Das Senden von *OLE_CLOSED* veranlaßt die Server-Bibliothek, Daten an die Client-Bibliothek zu übergeben. Es ist deshalb nicht notwendig, *OLE_CHANGED* oder *OLE_SAVED* zu senden. Wird die Rückfrage nicht bestätigt, sollte die Server-Anwendung nur die Funktion *OleRevokeServerDoc* aufrufen (ohne die Benachrichtigung *OLE_CLOSED* zu senden).

Mit der Funktion *OleRevokeObject* kann eine Server-Anwendung den Zugriff einer Client-Anwendung auf ein Objekt aufheben – z.B. wenn der Benutzer das Objekt freigibt und entfernt. In ähnlicher Form hebt die Funktion *OleRevokeServerDoc* den Zugriff auf ein Dokument für eine Client-Anwendung auf. Da die Funktion *OleRevokeServerDoc* die Zugriffsmöglichkeit des Client für alle Objekte in einem Dokument aufhebt, muß eine Anwendung nicht die Funktion *OleRevokeObject* für Objekte in diesem Dokument aufrufen. Mit der Funktion *OleRevokeServer* kann die Server-Anwendung alle Konversationen mit Client-Anwendungen beenden. Diese Funktionen teilen der Server-Bibliothek mit, daß die angegebenen Elemente nicht länger verfügbar sind.

Eine Server-Anwendung kann die Botschaft *OLE_WAIT_FOR_RELEASE* empfangen, z.B. kann die Funktion *OleRevokeServerDoc* diesen Wert zurückliefern. Obwohl eine Server-Anwendung in einer Botschaftsschleife auf den Aufruf der Funktion *Release* durch die Bibliothek warten kann, sollten Server niemals innerhalb von Callback-Funktionen in Botschaftsschleifen eintreten, die von ihr selbst bereitgestellt und von der Server-Bibliothek aufgerufen werden.

Die Client-Anwendung sollte die Server-Anwendung nie anweisen, das Dokument zu schließen oder die Ausführung zu beenden, wenn die Server-Anwendung ein verknüpftes Objekt bearbeitet. Eine Ausnahme besteht nur für den Fall, daß die Server-Anwendung die Verknüpfung aktualisiert, ohne dem Benutzer das Objekt anzuzeigen. Da ein verknüpftes Objekt unabhängig von der Client-Anwendung existiert, steuert der Benutzer mit der Server-Anwendung das Speichern und Schließen des Dokuments.

Ist einer Server-Anwendung beim Schließen die Zwischenablage zugeordnet, sollte sie sich vergewissern, daß die Daten vollständig und in der richtigen Reihenfolge in der Zwischenablage stehen. So sollten z.B. alle Native-Daten mit dem OwnerLink-Format verbunden sein.

Objekt-Handler

Durch die Verwendung eines Objekt-Handler kann ein Anwendungsentwickler auf den jeweiligen Bedarf zugeschnittene Eigenschaften bei der OLE-Imple-

mentierung bereitstellen. Existiert für eine Objektklasse ein Objekt-Handler, übernimmt dieser einige oder alle Funktionen, die ansonsten von der Client-Bibliothek und der Server-Bibliothek aufgerufen werden. Dabei kann der Objekt-Handler für jede der von ihm abgefangenen Funktionen besondere Aktionen ausführen. Der Objekt-Handler übergibt Funktionen, für die er keine Aktionen ausführt, an die Client-Bibliothek, die dann die Standardbearbeitung für diese Klasse implementiert.

Eine Anwendung kann anstelle von Metafiles und Bitmaps einen Objekt-Handler benutzen, um Native-Daten als Präsentationsdaten für ein Objekt zu übergeben. Außerdem kann mit einem Objekt-Handler eine spezielle Vorgehensweise beim Öffnen eines Objekts implementiert werden.

Objekt-Handler implementieren

Eine Server-Anwendung, die einen Objekt-Handler installiert, registriert den Handler mit dem Schlüsselwort handler für die Registrierungsdatenbank. Sobald eine Client-Anwendung eine der Funktionen zur Erzeugung von Objekten aufruft, benutzt die Client-Bibliothek den für das Objekt angegebenen Klassennamen und das Schlüsselwort handler, um die Registrierungsdatenbank zu durchsuchen. Findet die Bibliothek einen Objekt-Handler, lädt sie ihn und ruft ihn auf, um das Objekt zu erzeugen. Der Objekt-Handler kann entweder mit seinen eigenen objekterzeugenden Funktionen und Methoden ein Objekt erzeugen oder die dazu von der Client-Bibliothek bereitgestellten Standardfunktionen aufrufen.

Die Client-Bibliothek exportiert objekterzeugende OLE-Funktionen mit neuen Namen. Dabei wird das Präfix *Ole* immer durch *Def* (*default* für Voreinstellung) ersetzt. Objekt-Handler können diese Funktionen importieren und beim Erzeugen von Objekten verwenden.

Die folgenden Funktionen müssen von Objekt-Handlers importiert werden:

OLE-Funktion	Exportierter Name der Client-Bibliothek
OleCreate	*DefCreate*
OleCreateFromClip	*DefCreateFromClip*
OleCreateFromFile	*DefCreateFromFile*
OleCreateFromTemplate	*DefCreateFromTemplate*
OleCreateLinkFromClip	*DefCreateLinkFromClip*
OleCreateLinkFromFile	*DefCreateLinkFromFile*
OleLoadFromStream	*DefLoadFromStream*

Wenn ein Objekt-Handler eine Funktion definiert, die von der Client-Anwendung aufgerufen werden soll, sollte er den Namen der entsprechenden Ole-Funktion benutzen und nur »Ole« durch das Präfix »Dll« ersetzen. Verwendet ein Objekt-Handler z.B. die Funktion *DefCreate*, die von der Client-Bibliothek exportiert wurde, sollt er sie in *DllCreate* umbenennen. Wenn die Client-Bibliothek für eine Objektklasse einen Objekt-Handler findet, ruft sie durch Angabe

des Präfix »Dll« die handler-spezifischen Funktionen zur Erzeugung von Objekten auf.

Ruft der Handler eine der Standardfunktionen zur Erzeugung von Objekten auf, erhält er das Handle einer *OLEOBJECT*-Struktur, die auf die *OLEOBJECTVTBL*-Struktur zeigt, in der die aktuellen Verwaltungsfunktionen für Objekte enthalten sind. Der Objekt-Handler sollte die *OLEOBJECTVTBL*-Struktur kopieren und anpassen, indem er alle Funktionszeiger der Struktur durch Zeiger auf eigene Funktionen ersetzt. Speichert der Objekt-Handler die Zeiger auf die Standardfunktionen, kann auch jede der ersetzten Funktionen die Standardfunktionen in der Tabelle der Funktionszeiger aufrufen. Nachdem der Objekt-Handler die Anpassung der Struktur beendet hat, sollte der Zeiger auf die alte *OLEOBJECTVTBL*-Struktur durch einen Zeiger auf die geänderte *OLEOBJECTVTBL*-Struktur ersetzt werden.

Ruft die Client-Anwendung eine Funktion in der Client-Bibliothek auf, wird der Aufruf über die *OLEOBJECTVTBL*-Struktur des Objekt-Handler weitergegeben. Dabei wird der Aufruf an die vom Handler bereitgestellte Funktion weitergeleitet, wenn der Funktionszeiger vom Handler ersetzt wurde. Andernfalls wird der Aufruf an die Client-Bibliothek weitergeleitet.

Jede der Strukturen *OLECLIENT*, *OLEOBJECT*, *OLESERVER*, *OLESERVERDOC* und *OLESTREAM* enthält einen Zeiger auf eine Struktur, die eine Tabelle mit Funktionszeigern enthält. Strukturen, die Tabellen mit Funktionszeigern enthalten, sind an dem Suffix »VTBL« erkennbar. Jede der Strukturen, die einen Zeiger auf eine »VTBL«-Struktur enthält, kann außerdem zusätzliche instanzenspezifische Informationen enthalten. Diese Informationen sind nur für die Anwendung von Bedeutung, die die jeweilige Struktur bereitstellt, und sollten von anderen Anwendungen nicht benutzt werden. So sollte z.B. ein Objekt-Handler nicht versuchen, instanzenspezifische Informationen in einer *OLECLIENT*-Struktur zu benutzen.

Der Objekt-Handler sollte bei der Definition spezialisierter Funktionen die Konventionen zur Umbenennung (*Def* und *Dll*) verwenden. Wenn z.B. ein Objekt-Handler die Funktion *Draw* aus der *OLEOBJECTVTBL*-Struktur eines Objekts abändert, sollte er diese Funktion in eine Funktion *DefDraw* kopieren und *Draw* durch eine spezialisierte Funktion *DllDraw* ersetzen. Innerhalb der Funktion *DllDraw* kann der Objekt-Handler die Funktion *DefDraw* aufrufen, falls für einen bestimmten Fall die Standardzeichenoperation geeignet ist.

Das folgende Beispiel demonstriert, wie Zeiger auf Funktionen kopiert und ersetzt werden. Funktionen mit dem Präfix »Dll« sollten in der Moduldefinitionsdatei exportiert werden.

```
/* Deklaration der Funktionen DllDraw und DefDraw. */

OLESTATUS FAR PASCAL DllDraw(LPOLEOBJECT, HDC, LPRECT, LPRECT, HDC);
OLESTATUS (FAR PASCAL *DefDraw)(LPOLEOBJECT, HDC, LPRECT, LPRECT,
   HDC);

/* Kopieren der Funktion Draw aus OLEOBJECTVTBL nach DefDraw. */

    DefDraw = lpobj->lpvtbl->Draw;

/* DllDraw in OLEOBJECTVTBL kopieren. */

    *lpobj->lpvtbl->Draw = DllDraw;
```

```
OLESTATUS FAR PASCAL DllDraw(lpObject, hdc, lpBounds,
        lpWBounds, hdcFormat)
LPOLEOBJECT lpObject;
HDC         hdc;
LPRECT      lpBounds;
LPRECT      lpWBounds;
HDC         hdcFormat;
{
  /* DefDraw zurückgeben, falls keine Native-Daten verfügbar sind */

  if ((*lpobj->lpvtbl->GetData) (lpobj, cfNative, &hData) !=OLE_OK)
      return (*DefDraw) (lpobj, hdc, lpBounds, lpWBounds,
              hdcFormat);
  .
  .
  .
}
```

Objekte mit einem Objekt-Handler erzeugen

Die meisten der OLE-Funktionen zum Erzeugen von Objekten arbeiten identisch, wenn sie unbenannt und von der DLL eines Objekt-Handler benutzt werden. Ausnahmen sind nur die beiden Funktionen *OleCreateFromClip* und *OleLoadFromStream*.

DefCreateFromClip und DllCreateFromClip

Wenn die Client-Bibliothek die Funktion *DllCreateFromClip* aufruft, fügt sie einen zusätzlichen Parameter ein, der beim Aufruf der Funktion *OleCreateFromClip* nicht angegeben wird. Dieser Parameter, *objtype*, gibt an, ob das zu erzeugende Objekt ein eingebettetes oder ein verknüpftes Objekt ist. Sein Wert kann entweder *OT_LINKED* oder *OT_EMBEDDED* sein.

Der folgende Syntaxblock zeigt den Parameter *objtype*, wenn ein Objekt-Handler die Funktion *DefCreateFromClip* benutzt. Die Syntax der Funktion *DllCreateFromClip* ist identisch mit der Syntax der Funktion *DefCreateFromClip*.

```
OLESTATUS DefCreateFromClip(lpszProtocol, lpClient, lhClientDoc,
    lpszObjname, lplpObject, renderopt, cfFormat, objtype);
LPSTR lpszProtocol;          /* Adresse des String für den
                                Protokollnamen                  */
LPOLECLIENT lpClient;        /* Adresse der Client-Struktur     */
LHCLIENTDOC lhClientDoc;     /* long-Handle des Client-Dokuments */
LPSTR lpszObjname;           /* String für den Objektnamen      */
LPOLEOBJECT FAR *lplpObject; /* Adresse des Zeigers auf das
                                Objekt                          */
OLEOPT_RENDER renderopt;     /* übergabeoptionen                */
OLECLIPFORMAT cfFormat;      /* Zwischenablageformat            */
LONG objtype;                /* OT_LINKED oder OT_EMBEDDED      */
```

Ruft die Funktion *DllCreateFromClip* die Funktion *DefCreateFromClip* auf, sollte sie den Parameter *objtype* zusammen mit den anderen Parametern der Version der Funktion *DefCreateFromClip* übergeben, die von der Client-Bibliothek exportiert wurde. Dabei kann die Funktion *DllCreateFromClip* einige dieser Parameter ändern, bevor sie sie an die Funktion *DefCreateFromClip* weitergibt. So könnte z.B. der Objekt-Handler beim Aufruf der Funktion *DefCreateFromClip* einen anderen Wert für den Parameter *renderopt* angeben. Wenn die Client-Anwendung diese Funktion mit dem Wert *olerender_draw* für den Parameter *renderopt* aufruft und der Handler das Zeichnen mit Native-Daten durchführt, könnte der Handler den Wert *olerender_draw* in *olerender_none* ändern. Ruft der Handler jedoch die Funktion *GetData* auf und führt das Zeichnen basierend auf einem klassenspezifischen Format aus, so könnte er den Wert *olerender_draw* in *olerender_format* ändern. Benötigt der Handler ein anderes Übergabeformat als das von der Client-Anwendung angegebene, könnte er auch den Wert für den Parameter *cfFormat* im Aufruf der Funktion *DefCreateFromClip* ändern.

Verwendet ein Objekt-Handler Native-Daten, um ein eingebettetes Objekt zu übergeben, kann er die Bibliothek aufrufen und *olerender_none* angeben. Benutzt der Handler Native-Daten, um ein verknüpftes Objekt zu übergeben, kann er *olerender_format* benutzen und Native-Daten angeben. Wird die Funktion *Draw* des Handler aufgerufen, ruft der Handler die Funktion *GetData* auf und bestimmt Native-Daten für die Übergabe. Benutzt der Handler ein privates Datenformat, verläuft die Prozedur entsprechend, mit der Ausnahme, daß das private Format mit der Option *olerender_format* und der Funktion *GetData* angegeben wird.

DefLoadFromStream und DllLoadFromStream

Wenn die Client-Bibliothek die Funktion *DllLoadFromStream* aufruft, fügt die Bibliothek drei Parameter hinzu, die beim Aufruf der Funktion *OleLoadFromStream* nicht angegeben werden. Einer der Parameter ist der bereits für die Funktionen *DefCreateFromClip* und *DllCreateFromClip* beschriebene Parameter *objtype*. Die zwei anderen Parameter sind *aClass*, ein Atom, das den Klassennamen für das Objekt enthält und *cfFormat*, der ein privates Format für die Zwischenablage angibt, das der Objekt-Handler zur Übergabe des Objekts benutzen kann.

Der folgende Syntaxblock zeigt die Parameter *objtype*, *aClass* und *cfFormat*, wenn ein Objekt-Handler die Funktion *DefLoadFromStream* benutzt. Die Syntax der Funktion *DllLoadFromStream* ist identisch mit der Syntax der Funktion *DefLoadFromStream*.

```
OLESTATUS DefLoadFromStream(lpStream, lpszProtocol, lpClient,
       lhClientDoc, lpszObjname, lplpObject, objtype, aClass,
       cfFormat);
LPOLESTREAM lpStream;        /* Adresse des Stream für das Objekt  */
LPSTR lpszProtocol;          /* Adresse des String für den
                                Protokollnamen                     */
LPOLECLIENT lpClient;        /* Adresse der Client-Struktur        */
LHCLIENTDOC lhClientDoc;     /* long-Handle des Client-Dokuments   */
LPSTR lpszObjname;           /* String für den Objektnamen         */
```

```
LPOLEOBJECT FAR *lplpObject; /* Adresse des Zeigers auf
                                das Objekt                    */
LONG objtype;                /* OT_LINKED oder OT_EMBEDDED    */
ATOM aClass;                 /* Atom mit dem Klassennamen des
                                Objekts                       */
OLECLIPFORMAT cfFormat;      /* privates übergabeformat       */
```

Ruft die Funktion *DllLoadFromStream* die Funktion *DefLoadFromStream* auf, sollte sie die drei zusätzlichen Parameter zusammen mit den anderen Parametern der Version der Funktion *DefLoadFromStream* übergeben, die von der Client-Bibliothek exportiert wurde.

Dabei kann die Funktion *DllLoadFromStream* einige dieser Parameter ändern, bevor sie an die Funktion *DefLoadFromStream* weitergegeben werden. Der Objekt-Handler könnte z.B. den Wert für den Parameter *cfFormat* ändern, um ein privates Datenformat anzugeben, mit dem er das Objekt übergibt.

Wenn die Client-Anwendung den Objekt-Handler mit der Funktion *DefLoadFromStream* aufruft, erhält der Handler mit der Funktion *Get* der *OLESTREAMVTBL*-Struktur die Daten für das Objekt.

DDE direkt anwenden

Die OLE-Bibliotheken OLECLI.DLL und OLESVR.DLL benutzen DDE-Botschaften, um miteinander zu kommunizieren. Obwohl Client- und Server-Anwendungen DDE direkt benutzen können, ohne OLECLI.DLL oder OLESVR.DLL zwischenzuschalten, ist diese Methode der OLE-Implementierung nicht zu empfehlen. Zukünftige Erweiterungen der OLE-Bibliotheken dienen nur solchen Anwendungen, die die Bibliotheken verwenden, jedoch werden sie keine Anwendungen unterstützen, die DDE direkt benutzen.

Die folgenden Informationen über das auf DDE basierende OLE-Protokoll sind für Anwendungen gedacht, die DDE direkt implementieren müssen, obwohl dadurch die künftigen Erweiterungen des Systems nicht in Anspruch genommen werden können.

Das Implementieren des OLE-Protokolls erfordert das Implementieren des zugrundeliegenden DDE-Protokolls. Alle DDE-Standardregeln und -fähigkeiten sind gültig. Anwendungen, die diesem Protokoll entsprechen, müssen auch der DDE-Spezifikation entsprechen. Dies schließt auch ein, daß die Anwendungen das Thema System und die Standardelemente dieses Themas unterstützen müssen.

Client-Anwendungen und DDE

Beim Öffnen eines verknüpften oder eingebetteten Dokuments sollte die Client-Anwendung (wie im Abschnitt »Registrierung« beschrieben) den Klassennamen in der Registrierungsdatenbank suchen. Der folgende Pseudocode

illustriert die Abfolge der Ereignisse, wenn eine Client-Anwendung OLE durch DDE implementiert. Sobald eine Client-Anwendung eine Konversation mit einer Server-Anwendung aufbaut und von mehr als einer Server-Anwendung Antwort erhält, sollte sie die Antwort der ersten Server-Anwendung akzeptieren und die anderen verwerfen.

Verknüpftes Objekt:

```
WM_DDE_INITIATE Klassenname, Dokumentname
if not found {
    WM_DDE_INITIATE Klassenname, OLESystem
    if not found {
        WM_DDE_INITIATE Klassenname, System
        if not found {
           Start Anwendungsname, /Embedding
           fLaunched = true
           WM_DDE_INITIATE Klassenname, OLESystem
           if not found {
                 WM_DDE_INITIATE Klassenname, System
                 if not found
                        return error
           }
         }
     }

     /* Nun besteht eine Konversation mit dem Server
      * über das Thema System oder OLESystem. */

    WM_DDE_EXECUTE StdOpenDocument(DocumentName)
    WM_DDE_INITIATE Klassenname, Dokumentname
    if not found {
           if(fLaunched) WM_DDE_EXECUTE StdExit /* Ende */
                       return error
    }
}

/* Nun gibt es eine Konversation mit dem richtigen Dokument.*/
```

Eingebettetes Objekt:

```
WM_DDE_INITIATE Klassenname, OLESystem
if not found {
WM_DDE_INITIATE Klassenname, System
if not found {
   Start Anwendungsname, /Embedding
   fLaunched = true
   WM_DDE_INITIATE Klassenname, OLESystem
   if not found {
       WM_DDE_INITIATE Klassenname, System
       if not found
              return error
       }
```

```
    }
}

/* Nun besteht eine Konversation mit dem Server
 * über das Thema System oder OLESystem. */

 DDE_EXECUTE StdEditDocument(DocumentName)

/* Oder StdCreateDoc, falls es sich um den Befehl Objekt
 * einfügen handelt.*/

WM_DDE_INITIATE Klassenname, Dokumentname
if not found
  if(fLaunched) DDE_EXECUTE StdExit         /* Ende */
          return error
}

/*
 * Nun gibt es eine Konversation mit dem richtigen Dokument.
 */
```

Server-Anwendungen und DDE

Empfängt eine Server-Anwendung das Argument */Embedding* in der Kommandozeile, sollte sie kein neues Standarddokument erzeugen. Stattdessen sollte sie warten, bis die Client-Anwendung entweder den Befehl *StdOpenDocument* oder *StdEditDocument*, gefolgt von den Native-Daten, übergibt und die Server-Anwendung beauftragt, das Fenster anzuzeigen. Mit dem Element *StdHostNames* kann die Server-Anwendung den Namen der Client-Anwendung im Fenstertitel anzeigen.

Der folgende Pseudocode illustriert die Folge von Ereignissen, wenn eine Server-Anwendung OLE durch DDE implementiert. In diesem Beispiel sind zwei Fälle dargestellt. Ein Fall, in dem der Server eine einzelne Instanz verwendet, um alle Objekte zu bearbeiten (in untergeordneten MDI-Fenstern) und ein anderer Fall, in dem für jedes Objekt eine neue Instanz verwendet wird. Anwendungen, die für jedes Objekt eine neue Instanz benutzen und die bereits ein Dokument geöffnet haben, sollten Anforderungen zum Öffnen oder Erzeugen eines neuen Dokuments ablehnen.

```
MDI-Anwendung:

case WM_DDE_INITIATE:
   if Klassenname == diese Klasse {
      if (DocumentName == OLESystem || DocumentName == System)
         WM_DDE_ACK
      else if DocumentName == Name eines offenen Dokuments
         WM_DDE_ACK
   }
```

```
Mehrfachinstanzen-Anwendung:

case   WM_DDE_INITIATE:
    if  Klassenname == diese Klasse {
        if  (DocumentName == OLESystem ||  DocumentName  ==  Sys
            if  keine Dokumente ge´ffnet
                        WM_DDE_ACK
        }
        else if  DocumentName  ==  Name  eines  offenen  Dokument
                  WM_DDE_ACK
    }
```

Konversationen

Die Bearbeitung von Dokumenten findet während einer Konversation über das Thema System oder OLESystem einer Anwendung statt. Die Konversation wird mit dem Klassennamen des Dokuments eingerichtet.

Datentransfer und sonstige Operationen finden während der Konversationen mit dem Dokument (d.h. mit dem Thema) statt. Dabei wird die Konversation mit Hilfe des Dokumentnamens hergestellt.

Beachten Sie, daß der Themenname nur bei Initialisierungen von Konversationen verwendet wird und sich während der Konversation ändern kann. Die Möglichkeit der Umbenennung des Dokuments bedeutet nicht, daß zwei Namen für ein Dokument existieren können. Es ist somit sinnvoll, den Themennamen mit dem Dokumentnamen zu verbinden.

Elemente des Themas System

Einer Anwendung, die OLE unmittelbar über DDE benutzt, stehen drei neue Elemente für das Thema System zur Verfügung: Topics, Protocols und Status.

Das Element Topics liefert eine Liste der DDE-Themennamen zurück, die die Server-Anwendung geöffnet hat. Entsprechen die Themen Dokumenten, ist der Themenname zugleich der Dokumentname.

Das Element Protocols liefert eine Liste der Protokollnamen zurück, die von der Anwendung unterstützt werden. Die einzelnen Protokollnamen in dieser Liste sind durch Tabulatorzeichen voneinander getrennt. Ein Protokoll ist eine definierte Menge ausführbarer DDE-Strings, Elemente und Formatkonventionen, die die Anwendung benutzen kann. Das derzeit für verknüpfte und eingebettete Objekte definierte Protokoll ist wie folgt aufgebaut:

```
Protocols:     StdFileEditing/Elemente/Formate
```

Um mit Client-Anwendungen kompatibel zu sein, die vor der Implementierung des OLE-Protokolls geschrieben wurden, sollten Server-Anwendungen , die das DDE-Protokoll direkt benutzen, auch den String "Embedding" in die Liste der Protokolle aufnehmen.

Das Element Status ist ein Textelement, das den Wert *Ready* zurückliefert,

wenn die Server-Anwendung bereit ist, auf DDE-Anforderungen zu antworten. Andernfalls wird *Busy* zurückgeliefert. Dieses Element kann abgefragt werden, um zu ermitteln, ob die Client-Anwendung Funktionen wie die zum Aktualisieren von Objekten anbieten soll. Da eine Server-Anwendung eine Anforderung verwerfen oder zurückstellen kann (auch, wenn das Element Status *Ready* zurückliefert), sollte der weitere Ablauf von Client-Anwendungen nicht nur von diesem Element abhängen.

Standardnamen für Elemente und die Steuerung von Benachrichtigungen

Anwendungen, die OLE durch direkte Verwendung des dynamischen Datenaustauschs (DDE) unterstützen, benutzen zu den regulären Daten- und Bildformaten weitere vier Formate der Zwischenablage, nämlich das ObjectLink-, OwnerLink-, Native- und Binär-Format. Das Binär-Format ist eine Byte-Folge, deren Interpretation implizit im Element enthalten ist. Die Elemente *EditEnumItems*, *StdTargetDevice* und *StdHostNames* sind im Binär-Format. Die ObjectLink-, OwnerLink-, und Native-Formate sind im Abschnitt »Konventionen der Zwischenablage« beschrieben. Für dieses Protokoll sind neue Elemente definiert, die für jedes Thema außer System zur Verfügung stehen:

StdDocumentName: Enthält den permanenten Dokumentnamen, der mit dem Thema verbunden ist. Ist kein permanenter Speicherplatz mit dem Thema verbunden, ist dieses Element leer. Das Element unterstützt Request- und Advise-Transaktionen. Außerdem kann es benutzt werden, um festzustellen, ob ein geöffnetes Dokument umbenannt wurde.

EditEnvItems: Liefert eine Liste der Elemente zurück, die Informationen über die Umgebung enthalten, die von der Server-Anwendung für ihre Dokumente unterstützt wird. Die Liste liegt in einem Textformat vor. Die einzelnen Einträge sind durch Tabulatoren voneinander getrennt. Derzeit definierte Elemente sind *StdHostNames*, *StdDocDimensions* und *StdTargetDevice*. Anwendungen können weitere Elemente deklarieren (und deren Interpretationen definieren, falls das Binär-Format benutzt wird), so daß die Client-Anwendungen, denen diese Elemente bekannt sind, detaillierte Informationen erhalten. Server-Anwendungen, die bestimmte Elemente nicht verwenden können, sollten deren Namen aus dem Element *EditEnvItems* streichen. Client-Anwendungen sollten dieses Element zusammen mit der Botschaft *WM_DDE_REQUEST* benutzen, um zu ermitteln, welche Elemente die Server-Anwendung verwenden kann. Sie sollten die Daten mit der Botschaft *WM_DDE_POKE* bereitstellen.

StdHostNames: Akzeptiert Informationen über die Client-Anwendung, die im Binär-Format als folgende Struktur interpretiert werden:

```
struct {
   WORD clientNameOffset;
   WORD documentNameOffset;
   BYTE data[];
} StdHostNames;
```

Die Offset-Werte sind relativ zum Beginn des Daten-Array. Sie geben den Startpunkt der jeweiligen Informationen im Array an.

StdTargetDevice: Akzeptiert Information über das Zielgerät, das die Client-Anwendung benutzt. Diese Information liegt im Binär-Format vor und wird als folgende Struktur interpretiert. Die Offset-Werte sind relativ zum Beginn des Daten-Array:

```
typedef struct _OLETARGETDEVICE {
   WORD otdDeviceNameOffset;
   WORD otdDriverNameOffset;
   WORD otdPortNameOffset;
   WORD otdExtDevmodeOffset;
   WORD otdExtDevmodeSize;
   WORD otdEnvironmentOffset;
   WORD otdEnvironmentSize;
   BYTE otdDate[];
} OLETARGETDEVICE;
```

StdDocDimensions: Akzeptiert Informationen über die Dokumentgröße. Die Informationen liegen im Binär-Format vor und werden als folgende Struktur interpretiert. Die Werte sind in *MM_HIMETRIC*-Einheiten angegeben.

```
struct {
   int iXContainer;
   int iYContainer;
} StdDocDimensions;
```

StdColorScheme: Liefert die Farben zurück, die die Server-Anwendung aktuell benutzt und akzeptiert Information über die Farben, die der Server-Anwendung von der Client-Anwendung vorgeschlagen werden. Diese Information besitzt Binär-Format und wird als eine *LOGPALETTE*-Struktur interpretiert.

Null: Bestimmt eine Request- oder Advise-Transaktion für alle Daten im Thema. Dieses Element ist ein Name der Länge Null.

Die Aktualisierung für Advise-Transaktionen über Elemente folgt einer Konvention, nach der dem tatsächlichen Namen des Elements ein Aktualisierungsbezeichner hinzugefügt wird.

```
Elementname\Aktualisierungstyp
```

Um mit früheren Versionen kompatibel zu sein, ist das Weglassen des Typs der Aktualisierung identisch mit der Angabe von */Change*. Der Platzhalter *Aktualisierungstyp* kann durch einen der folgenden Werte ersetzt werden.

Wert	Bedeutung
/Change	Meldet jede Änderung.
/Close	Meldet, wenn das Dokument geschlossen wird.
/Save	Meldet, wenn das Dokument gesichert wird.

DDE-Server-Anwendungen müssen jedes Auftreten einer *WM_DDE_ADVISE*-Botschaft speichern, die eine eindeutige Kombination aus *Elementname, Aktua-*

lisierungstyp, *Format* und *Konversation* angibt. Eine Benachrichtigung wird mit einer *WM_DDE_UNADVISE*-Botschaft mit entsprechenden Parametern außer Kraft gesetzt. Nach dem FIFO-Prinzip (first in, first out) wird die älteste Benachrichtigung als erste deaktiviert, wenn die *WM_DDE_UNADVISE*-Botschaft kein Format angibt.

Standardbefehle in ausführbaren DDE-Strings

Die Syntax für Standardbefehle, die in ausführbaren Strings übermittelt werden, ist identisch mit der Syntax für andere DDE-Befehle:

```
Befehl1 (Argument11, Argument12, ...) [Befehl2 (Argument21,
Argument22, ...)]
```

Bei Befehlen ohne Argumente sind keine Klammern erforderlich. String-

Argumente müssen in doppelte Klammern eingeschlossen sein.

Internationale Execute-Befehle

Ausführbare DDE-Strings werden normalerweise von einer Makrosprache in einer externen Anwendung übergeben und sind den jeweiligen länderspezifischen Konventionen angepaßt. Dagegen werden ausführbare OLE-Befehle von Anwendungsprogammen nur für den Eigenbedarf übersendet, müssen daher nicht übersetzt oder konvertiert werden. Sie müssen allerdings allgemein erkannt werden.

Die ausführbaren OLE-Standardbefehle sollten nicht übersetzt werden. Es werden Rechtschreibung und Trennzeichen der USA benutzt. Daher gelten folgende Regeln:

- Die Client-Anwendungen und die Client-Bibliothek übersenden die ausführbaren Standardbefehle im amerikanischen Format.
- Die Server-Bibliothek muß diese Befehle im amerikanischen Format erhalten.
- Server-Anwendungen, die unmittelbar das DDE-Protokoll unterstützen, sollten ausführbare DDE-Strings im amerikanischen Format analysieren, wenn sie selbst keine weiteren Befehle hinzufügen.
- Server-Anwendungen, die sowohl ausführbare OLE- als auch ausführbare länderspezifisch angepaßte DDE-Befehle unterstützen, sollten zunächst den String gemäß den länderspezifischen Trennzeichen analysieren. Mißlingt dies, sollten sie den String erneut im amerikanischen Format analysieren. Gelingt dies, sollten sie den Befehl ausführen. Wird der Befehl im amerikanischen Format empfangen, kann die Server-Anwendung zusätzlich prüfen, ob es sich bei diesem Befehl um einen gültigen Standardbefehl handelt.

Notwendige Befehle

In diesem Abschnitt werden Befehle aufgelistet, die von Server-Anwendungen unterstützt werden müssen. Die Befehle *StdNewDocument*, *StdNewFromTemplate*, *StdEditDocument* und *StdOpenDocument* stellen das Dokument für

DDE-Konversationen mit dem Namen *Dokumentname* bereit. Diese Funktionen blenden jedoch kein mit dem Dokument verbundenes Fenster ein. Um das Fenster sichtbar zu machen, muß die Client-Anwendung entweder gemeinsam die beiden Befehle *StdShowItem* und *StdDoVerbItem* oder aber nur den Befehl *StdDoVerbItem* alleine übersenden. Dadurch kann die Client-Anwendung zusätzliche Parameter mit der Server-Anwendung abstimmen (wie z.B. das Element *StdTargetDevice*) ohne ein unnötiges Neuzeichnen auszulösen.

StdNewDocument(Klassenname, Dokumentname) Erzeugt ein neues, leeres Dokument der angegebenen Klasse und mit dem angegebenen Dokumentnamen, speichert das Dokument jedoch nicht ab. Die Server-Anwendung sollte einen Fehlerwert zurückliefern, falls der Dokumentname bereits existiert. Erhält die Client-Anwendung diese Fehlermeldung, sollte sie einen anderen Dokumentnamen erzeugen und eine erneute Ausführung der Funktion versuchen.

Die Server-Anwendung sollte dabei das Fenster solange nicht einblenden, bis sie einen *StdShowItem*-Befehl erhält. Das Warten auf den Befehl *StdShowItem* der Client-Anwendung ermöglicht es der Client-Anwendung zusätzliche Parameter (z.B. durch Verwenden von *StdTargetDevice*) mit der Server-Anwendung abzustimmen, ohne daß das Fenster neu gezeichnet werden muß.

StdNewFromTemplate(Klassenname, Dokumentname, Schablonen-Name) Erzeugt ein neues Dokument der angegebenen Klasse und mit dem angegebenen Dokumentnamen. Dazu wird die Schablone mit dem angegebenen permanenten Namen (entspricht dem Dateinamen) verwendet.

Die Server-Anwendung sollte dabei das Fenster solange nicht einblenden, bis sie den Befehl *StdShowItem* erhält. Das Warten auf den Befehl *StdShowItem* der Client-Anwendung ermöglicht es der Client-Anwendung zusätzliche Parameter (z.B. durch Verwenden von *StdTargetDevice*) mit der Server-Anwendung abzustimmen, ohne daß das Fenster neu gezeichnet werden muß.

StdEditDocument (Dokumentname) Erzeugt ein Dokument mitdem angegebenen Namen. Bereitet das Dokument darauf vor, Daten anzunehmen, die mit *WM_DDE_POKE* in das Dokument gebracht werden. Dabei sollte die Server-Anwendung eine Fehlermeldung zurückliefern, wenn der Dokumentname bereits existiert. Erhält die Client-Anwendung diese Fehlermeldung, sollte sie einen neuen Namen erzeugen und die Ausführung erneut versuchen.

Die Server-Anwendung sollte das Fenster solange nicht einblenden, solange sie den Befehl *StdShowItem* empfängt. Das Warten auf den Befehl *StdShowItem* der Client-Anwendung ermöglicht es der Client-Anwendung zusätzliche Parameter (z.B. durch Verwenden von *StdTargetDevice*) mit der Server-Anwendung abzustimmen, ohne daß das Fenster neu gezeichnet werden muß.

StdOpenDocument(Dokumentname) Wird an das Thema System gesendet. Mit diesem Befehl wird ein bereits existierendes Dokument mit dem angegebenen Namen geöffnet.

Die Server-Anwendung sollte das Fenster solange nicht einblenden, bis sie den Befehl *StdShowItem* erhält. Das Warten auf den Befehl *StdShowItem* der Client-Anwendung ermöglicht es der Client-Anwendung zusätzliche Parameter

(z.B. durch Verwenden von *StdTargetDevice*) mit der Server-Anwendung abzustimmen, ohne daß das Fenster neu gezeichnet werden muß.

StdCloseDocument(Dokumentname) Wird an das Thema System gesendet. Mit diesem Befehl wird das mit dem Dokument verbundene Fenster geschlossen. Zur Bestätigung beendet die Server-Anwendung jede Konversation, die mit dem Dokument verbunden ist. Während des Schließens sollte die Server-Anwendung das Fenster nicht aktivieren.

StdShowItem(Dokumentname, Elementname [,fDoNotTakeFocus]) Wird an das Thema System gesendet. Dieser Befehl macht das Fenster, das das genannte Dokument enthält, sichtbar und führt einen Bildlauf durch, um das genannte Element (falls vorhanden) anzuzeigen. Das optionale dritte Argument zeigt an, ob die Server-Anwendung den Fokus auf sich ziehen und selbst in den Vordergrund treten sollte. Dieses Argument sollte den Wert *TRUE* haben, falls die Server-Anwendung den Fokus nicht auf sich ziehen soll. Andernfalls sollte der Wert für dieses Argument *FALSE* sein. Der Standardwert ist *FALSE*.

StdExit Beendet die Server-Anwendung. Dieser Befehl sollte nur von der Client-Anwendung verwendet werden, die die Server-Anwendung gestartet hat. Dieser Befehl ist nur im Thema System verfügbar.

Der Befehl *StdExit* wird gesendet, um die Anwendung zu beenden, wenn es während der Startphase zu einem Fehler kommt oder wenn die Client-Anwendung die Server-Anwendung für eine Aktualisierung im Hintergrund gestartet hat. Falls Server-Anwendungen ungesicherte, vom Benutzer geöffnete, Daten besitzen, sollten sie diesen Befehl ignorieren.

Varianten der notwendigen Befehle

Die folgenden Abweichungen von den oben angeführten Befehlen können an das Dokumentthema anstatt an das Thema System gesendet werden. Damit kann eine Client-Anwendung, die bereits mit dem Dokument in Konversation steht, das Öffnen einer zusätzlichen Konversation mit dem System vermeiden. Der Dokumentname ist aus diesen Befehlen weggelassen, da er zum einen im Konversationsthema impliziert gegeben ist und zum anderen von der Server-Anwendung geändert worden sein kann. Eine solche Namensänderung beeinträchtigt die Konversation nicht. Jedoch muß die Server-Anwendung die Möglichkeit haben, mit den Konversationsinformationen das Objekt zu identifizieren, mit dem gearbeitet werden soll.

StdCloseDocument Wird an die Dokumentkonversation gesendet. Dieser Befehl schließt das mit der Konversation verbundene Dokument, ohne das zugehörige Fenster zu aktivieren. Dieser Befehl hat zur Folge, daß im Falle der Bestätigung das Fenster der Server-Anwendung eine *WM_DDE_TERMINATE*-Botschaft absendet.

StdDoVerbItem (Elementname, iVerb, fShow, fDoNotTakeFocus) Wird an die Dokumentkonversation gesendet. Dieser Befehl entspricht dem Befehl *StdShowItem*, mit der Ausnahme, daß er zusätzlich eine Integerzahl und ein Flag enthält. Die Integerzahl gibt an, welche der registrierten Operationen ausgeführt werden soll. Das Flag gibt an, ob das Fenster eingeblendet werden soll. Falls nötig, kann die Server-Anwendung das Flag *fShow* ignorieren.

StdShowItem (Elementname[,fDoNotTakeFocus]) Wird an die Dokumentkonversation gesendet. Dieser Befehl zeigt das Dokumentfenster auf dem Bildschirm und führt, falls nötig einen Bildlauf durch, um das Element in den sichtbaren Bereich zu bringen. Ist der Elementname *NULL*, wird kein Bildlauf durchgeführt. Das optionale zweite Argument gibt an, ob die Server-Anwendung den Fokus übernehmen und sich in den Vordergrund stellen soll. Dieses Argument sollte den Wert *TRUE* haben, wenn die Server-Anwendung den Fokus nicht übernimmt. Andernfalls hat es den Wert *FALSE*. Der Standardwert ist *FALSE*.

4. Windows-Funktionen

AbortDoc 3.1

Syntax

```
int AbortDoc(hdc)

HDC hdc;            /* Handle des Gerätekontexts     */
```

Beschreibung *AbortDoc* beendet den aktuellen Druckauftrag und löscht alle Ausgaben seit dem letzten Aufruf von *StartDoc*. Diese Funktion ersetzt den Drucker-Escape-Code *ABORTDOC*.

Parameter *hdc:* Bezeichnet den Gerätekontext für den Druckauftrag.

Rückgabewert War die Funktion erfolgreich, ist der Rückgabewert größer oder gleich Null, andernfalls ist er kleiner als Null.

Hinweis Anwendungen sollten die Funktion *AbortDoc* aufrufen, um Druckaufträge abzubrechen, wenn ein Fehler aufgetreten ist oder der Benutzer den Druckauftrag abbricht. Zum Beenden eines erfolgreichen Druckauftrags sollten Anwendungen die Funktion *EndDoc* benutzen. Wurde der Druck-Manager benutzt, um den Druckauftrag zu starten, löscht *AbortDoc* den gesamten Auftrag – der Drucker empfängt keine Daten. Andernfalls sind die Daten möglicherweise bereits dem Drucker übergeben worden, bevor *AbortDoc* aufgerufen wurde. Der Druckertreiber hat dann den Drucker zurückgesetzt und – falls möglich – den Druckauftrag geschlossen.

Siehe auch *EndDoc*, *SetAbortProc*, *StartDoc*

AbortProc 3.1

Syntax

```
BOOL CALLBACK AbortProc(hdc, error)

HDC hdc;            /* Handle des Gerätekontexts     */
int error;          /* Fehlerwert                    */
```

Beschreibung Die Funktion *AbortProc* ist eine anwendungsdefinierte Callback-Funktion. Sie wird aufgerufen, wenn ein Druckauftrag abgebrochen werden soll, während er in die Warteschlange geschrieben wird.

Parameter *hdc:* Bezeichnet den Gerätekontext.

error: Bestimmt, ob ein Fehler aufgetreten ist. Dieser Parameter ist Null, wenn kein Fehler aufgetreten ist. Er ist *SP_OUTOFDISK*, wenn der Druck-Manager aktuell keinen Plattenspeicher zur Verfügung hat, jedoch zusätzlicher Speicher verfügbar wird, wenn die Anwendung wartet. Hat dieser Parameter den Wert *SP_OUTOFDISK*, muß die Anwendung den Druckauftrag nicht abbrechen. Wenn die Anwendung den Druckauftrag nicht abbricht, muß sie die Steuerung an den Druck-Manager abgeben, indem sie die Funktion *PeekMessage* oder *GetMessage* aufruft.

Rückgabewert Die Callback-Funktion sollte *TRUE* zurückgeben, um den Druckauftrag fortzusetzen oder *FALSE*, um ihn abzubrechen.

Hinweis Anwendungen installieren diese Callback-Funktion durch Aufruf der Funktion *SetAbortProc*. *AbortProc* ist ein Platzhalter für den anwendungsdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS- Anweisung in der Moduldefinitionsdatei der Anwendung exportiert werden.

Siehe auch *GetMessage*, *PeekMessage*, *SetAbortProc*

AllocDiskSpace 3.1

Syntax

```
#include <stress.h>
int AllocDiskSpace(lLeft, uDrive)

long lLeft;     /* Anzahl noch verfügbarer Byte      */
UINT uDrive;    /* Festplattenpartition              */
```

Beschreibung *AllocDiskSpace* erzeugt eine Datei, die mindestens die angegebene Anzahl Byte auf der angegebenen Partition frei läßt. STRESS.EAT wird im Hauptverzeichnis der Partition erstellt.

Wenn STRESS.EAT beim Aufruf von *AllocDiskSpace* bereits existiert, löscht die Funktion die Datei und erstellt eine neue.

Parameter *lLeft:* Bestimmt die Anzahl Byte, die auf der Platte verfügbar bleiben müssen.

uDrive: Bestimmt die Partition, auf der die Datei STRESS.EAT erzeugt werden soll. Dieser Parameter muß einen der folgenden Werte haben:

Wert	Bedeutung
EDS_WIN	Erzeugt die Datei auf der Windows-Partition.
EDS_CUR	Erzeugt die Datei auf der aktuellen Partition.
EDS_TEMP	Erzeugt die Datei auf der Partition, die das in TEMP angegebene Verzeichnis enthält.

Rückgabewert Der Rückgabewert ist größer als Null, wenn die Funktion erfolgreich ist. Er ist Null, wenn die Funktion die Datei nicht erzeugen konnte. Der Rückgabewert ist -1, wenn mindestens einer der Parameter ungültig ist.

Hinweis In zwei Fällen kann die verbleibende Anzahl Byte auf der Platte kleiner sein kann als die im Parameter *lLeft* angegebene Anzahl: wenn die verfügbare Anzahl Byte auf der Platte bereits beim Aufruf der Funktion *AllocDiskSpace* kleiner ist, als die in *lLeft* angegebene Anzahl, oder wenn der Wert von *lLeft* kein Vielfaches der Cluster-Größe der Platte ist. *UnAllocDiskSpace* löscht die Datei, die von *AllocDiskSpace* erzeugt wird.

Siehe auch *UnAllocDiskSpace*

AllocFileHandles 3.1

Syntax
```
#include <stress.h>
int AllocFileHandles(Left)

int Left;       /* Anzahl der Datei-Handle,          */
```

Beschreibung *AllocFileHandles* weist Datei-Handle zu, bis nur noch die angegebene Anzahl Datei-Handle für die aktuelle Instanz der Anwendung verfügbar ist. Wenn beim Aufruf der Funktion *AllocFileHandles* genau diese oder eine kleinere Zahl Handle verfügbar ist, kehrt die Funktion unmittelbar zurück. Bevor neue Handles reserviert werden, gibt diese Funktion alle Handles frei, die zuvor mit *AllocFileHandles* zugewiesen wurden.

Parameter *Left:* Gibt die Anzahl der Datei-Handles an, die verfügbar bleiben sollen.

Rückgabewert Der Rückgabewert ist größer als Null, wenn *AllocFileHandles* mindestens ein Datei-Handle erfolgreich zuweisen konnte. Er ist Null, wenn weniger als die angegebene Anzahl Datei-Handles verfügbar waren, als *AllocFileHandles* aufgerufen wurde. Der Rückgabewert ist -1, wenn der Parameter *Left* negativ ist.

Hinweis *AllocFileHandles* weist nicht mehr als 256 Datei-Handles zu, unabhängig von der für die Anwendung verfügbaren Anzahl.

Die Funktion *UnAllocFileHandles* gibt alle Datei-Handles frei, die vorher durch *AllocFileHandles* reserviert wurden.

Siehe auch *UnAllocFileHandles*

AllocGDIMem 3.1

Syntax
```
#include <stress.h>
BOOL AllocGDIMem(uLeft)

UINT uLeft;          /* Anzahl verbleibender Byte   */
```

Beschreibung Die Funktion *AllocGDIMem* reserviert Speicher auf dem Heap der GDI (grafische Geräteschnittstelle), bis nur noch die angegebene Anzahl Byte verfügbar ist. Bevor die Speicherreservierungen vorgenommen werden, gibt diese Funktion den vorher mit *AllocGDIMem* reservierten Speicher frei.

Parameter *uLeft:* Bestimmt die Größe des Speichers in Byte, der auf dem GDI-Heap verfügbar bleiben soll.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *FreeAllGDIMem* gibt Speicher frei, der zuvor mit *AllocGDIMem* reserviert wurde.

Siehe auch *FreeAllGDIMem*

AllocMem 3.1

Syntax
```
#include <stress.h>
BOOL AllocMem(dwLeft)

DWORD dwLeft; /* kleinstmögliche Speicherreservierung */
```

Beschreibung Die Funktion *AllocMem* reserviert globalen Speicher, bis nur noch die angegebene Anzahl Byte auf dem globalen Heap verfügbar ist. Bevor neue Speicherreservierungen vorgenommen werden, gibt diese Funktion den Speicher frei, der zuvor durch *AllocMem* reserviert wurde.

Papameter *dwLeft:* Bestimmt die minimale Größe der vorzunehmenden Speicherreservierung in Byte.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *FreeAllMem* gibt den Speicher frei, der zuvor durch *AllocMem* reserviert wurde.

Siehe auch *FreeAllMem*

AllocUserMem 3.1

Syntax
```
#include <stress.h>
BOOL AllocUserMem(uContig)

UINT uContig; /* kleinstmögliche Speicherreservierung */
```

Beschreibung Die Funktion *AllocUserMem* reserviert Speicher auf dem USER-Heap, bis nur noch die angegebene Anzahl Byte verfügbar ist. Bevor diese Funktion neue Reservierungen durchführt, hebt sie zuvor durch *AllocUserMem* durchgeführte Reservierungen auf.

Parameter *uContig:* Gibt die kleinste Größe für die durchzuführende Speicherreservierung in Byte an.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *FreeAllUserMem* gibt den Speicher frei, der zuvor durch *AllocUserMem* reserviert wurde.

Siehe auch *FreeAllUserMem*

CallNextHookEx 3.1

Syntax

```
LRESULT CallNextHookEx(hHook, nCode, wParam, lParam)

HHOOK hHook;          /* Handle der Filterfunktion    */
int nCode;            /* Filter-Code                  */
WPARAM wParam;        /* erster Botschaftsparameter   */
LPARAM lParam;        /* zweiter Botschaftsparameter  */
```

Beschreibung Die Funktion *CallNextHookEx* übergibt die Filterinformationen an die nächste Filterfunktion in der Kette der Filterfunktionen.

Parameter *hHook:* Bezeichnet die aktuelle Filterfunktion.

nCode: Bestimmt den Filter-Code, der an die nächste Filterfunktion übergeben werden soll. Eine Filterfunktion benutzt diesen Code, um die Bearbeitung von Botschaften festzulegen, die an den Filter übergeben werden.

wParam: Bestimmt 16 Bits zusätzlicher, botschaftsabhängiger Informationen.

lParam: Bestimmt 32 Bits zusätzlicher, botschaftsabhängiger Informationen.

Rückgabewert Der Rückgabewert bestimmt das Ergebnis der Botschaftsbearbeitung und hängt vom Wert des Parameters *nCode* ab.

Hinweis Der Aufruf der Funktion *CallNextHookEx* ist optional. Anwendungen können diese Funktion vor oder nach Abschluß der Bearbeitung in ihrer eigenen Filterfunktion aufrufen. Wenn eine Anwendung *CallNextHookEx* nicht benutzt, ruft Windows keine Filterfunktionen auf, die bereits vor der Filterfunktion der Anwendung installiert wurden.

Siehe auch *SetWindowsHookEx, UnhookWindowsHookEx*

CallWndProc 3.1

Syntax

```
LRESULT CALLBACK CallWndProc(code, wParam, lParam)

int code;             /* Botschaftsbearbeitungs-Flag  */
WPARAM wParam;        /* aktuelles Task-Flag          */
```

```
LPARAM LParam;      /* Adresse der Struktur mit
                       den Botschaftsdaten          */
```

Beschreibung Die Funktion *CallWndProc* ist eine bibliotheksdefinierte Callback-Funktion, die vom System immer dann aufgerufen wird, wenn auch die Funktion *SendMessage* aufgerufen wird. Das System übergibt die Botschaft an die Callback-Funktion, bevor sie die Botschaft an die Fensterfunktion des Zielfensters übergibt.

Parameter *code:* Legt fest, ob die Callback-Funktion die Botschaft bearbeitet oder die Funktion *CallNextHookEx* aufrufen soll. Ist der Parameter *code* kleiner als Null, sollte die Callback-Funktion die Botschaft ohne weitere Bearbeitung an *CallNextHookEx* übergeben.

wParam: Bestimmt, ob die Botschaft von der aktuellen Task übergeben wird. In diesem Fall ist dieser Parameter ungleich Null, andernfalls ist er *NULL*.

lParam: Zeigt auf eine Struktur, die Details über die Botschaft enthält. Nachfolgend sehen Sie die Reihenfolge, den Typ und die Beschreibung jedes Elements dieser Struktur:

Element	Beschreibung
lParam	Enthält den Parameter *lParam* der Botschaft.
wParam	Enthält den Parameter *wParam* der Botschaft.
uMsg	Bestimmt die Botschaft.
hWnd	Bezeichnet das Fenster, das die Botschaft empfangen wird.

Rückgabewert Die Callback-Funktion sollte Null zurückgeben.

Hinweis Die Callback-Funktion *CallWndProc* kann die Botschaft wie gewünscht bearbeiten oder modifizieren. Nach Rückkehr der Funktion wird die Botschaft mit ihren Modifikationen an die Fensterfunktion übergeben.

Diese Callback-Funktion muß sich in einer dynamischen Linkbibliothek befinden.

Anwendungen müssen die Callback-Funktion installieren, indem sie als Filtertyp *WH_CALLWNDPROC* angeben und die Adresse der Prozedurinstanz der Callback-Funktion in einem Aufruf der Funktion *SetWindowsHookEx* übergeben.

CallWndProc ist ein Platzhalter für den bibliotheksdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Bibliothek exportiert werden.

Siehe auch *CallNextHookEx*, *SendMessage*, *SetWindowsHookEx*

CBTProc 3.1

Syntax

```
LRESULT CALLBACK CBTProc(code, wParam, lParam)

int code;            /* CBT-Filter-Code             */
WPARAM wParam;       /* Abhängig vom Parameter code */
LPARAM lParam;       /* Abhängig vom Parameter code */
```

Beschreibung

Die Funktion *CBTProc* ist eine bibliotheksdefinierte Callback-Funktion, die vom System aufgerufen wird, bevor ein Fenster aktiviert, erzeugt, freigegeben, zum Symbol verkleinert, zum Vollbild vergrößert, verschoben oder in seiner Größe verändert wird. Diese Funktion wird auch vor dem Beenden eines Systembefehls, vor dem Entfernen eines Maus- oder Tastaturereignisses aus der Botschaftswarteschlange des Systems, vor dem Setzen des Fokus oder vor der Synchronisierung mit der Botschaftswarteschlange des Systems aufgerufen.

Der zurückgegebene Wert der Callback-Funktion legt fest, ob eine dieser Operationen zugelassen oder verhindert werden soll.

Parameter

code: Bestimmt einen CBT-Filter-Code (computer-based-training), der die auszuführende Operation bezeichnet oder einen Wert kleiner Null, wenn die Callback-Funktion die Parameter *code*, *wParam* und *lParam* an die Funktion *CallNextHookEx* übergeben soll. Der Parameter *code* kann einen der folgenden Werte annehmen:

Code	Bedeutung
HCBT_ACTIVATE	Das System wird ein Fenster aktivieren.
HCBT_CLICK-SKIPPED	Das System hat eine Mausbotschaft aus der Botschaftswarteschlange des Systems entfernt. Eine CBT-Anwendung, die einen Aufzeichnungsprotokollfilter als Reaktion auf Mausbotschaften installieren muß, sollte dies tun, sobald sie diesen Filter-Code erhält.
HCBT_CREATE-WND	Ein Fenster soll erzeugt werden. Das System ruft die Callback-Funktion auf, bevor eine *WM_CREATE*- oder *WM_NCCREATE*-Botschaft an das Fenster übergeben wird. Wenn die Callback-Funktion *TRUE* zurückgibt, gibt das System das Fenster frei. In diesem Fall gibt die Funktion *CreateWindow NULL* zurück, die *WM_DESTROY*-Botschaft wird aber nicht an das Fenster übergeben. Gibt die Callback-Funktion *FALSE* zurück, wird das Fenster wie gewohnt erzeugt.

Code	Bedeutung
	Zum Zeitpunkt des Eintreffens der Benachrichtigung *HCBT_CREATEWND* wurde das Fenster bereits erzeugt, jedoch stehen die endgültige Größe und Position noch nicht fest. Außerdem wurde das übergeordnete Fenster noch nicht initialisiert. Es ist möglich, Botschaften an das neu erzeugte Fenster zu übergeben, obwohl es noch keine *WM_NCCREATE*- oder *WM_CREATE*-Botschaften erhalten hat. Es ist möglich, die Z-Ordnung des neu erzeugten Fensters zu ändern, indem das Element *hwndInsertAfter* der *CBT_CREATEWND*-Struktur geändert wird.
HCBT_DESTROYWND	Ein Fenster soll freigegeben und entfernt werden.
HCBT_KEYSKIPPED	Das System hat eine Tastaturbotschaft aus der Botschaftswarteschlange des Systems entfernt. Eine CBT-Anwendung, die einen Aufzeichnungsprotokollfilter als Reaktion auf Tastaturbotschaften installieren muß, sollte dies tun, sobald sie diesen Filter-Code empfängt.
HCBT_MINMAX	Ein Fenster soll zum Symbol verkleinert oder zum Vollbild vergrößert werden.
HCBT_MOVESIZE	Ein Fenster soll verschoben oder in seiner Größe verändert werden.
HCBT_QS	Das System hat eine *WM_QUEUESYNC*-Botschaft aus der Botschaftswarteschlange des Systems abgerufen.
HCBT_SETFOCUS	Ein Fenster soll den Fokus erhalten.
HCBT_SYSCOMMAND	Ein Systembefehl soll ausgeführt werden. Dies erlaubt es einer CBT-Anwendung, die Task-Umschaltung durch Tastenkürzel zu deaktivieren.

wParam: Dieser Parameter ist vom Parameter *code* abhängig. Details finden Sie im »Hinweis«.

lParam: Dieser Parameter ist vom Parameter *code* abhängig. Details finden Sie im »Hinweis«.

Rückgabewert Die Callback-Funktion sollte Null zurückgeben, um die Operationen zuzulassen, die den folgenden CBT-Filter-Codes entsprechen. Der Rückgabewert 1 unterbindet diese Operationen:

HCBT_ACTIVATE
HCBT_CREATEWND
HCBT_DESTROYWND
HCBT_MINMAX
HCBT_MOVESIZE
HCBT_SYSCOMMAND

Der Rückgabewert wird für alle Operationen ignoriert, die den folgenden CBT-Filter-Codes entsprechen:

HCBT_CLICKSKIPPED
HCBT_KEYSKIPPED
HCBT_QS

Hinweis

Die Callback-Funktion sollte einen Wiedergabeprotokollfilter nur für die oben beschriebenen Situationen installieren.

Diese Callback-Funktion muß sich in einer dynamischen Linkbibliothek befinden.

Anwendungen müssen die Callback-Funktion installieren, indem sie den Filtertyp *WH_CBT* und die Adresse der Prozeduriinstanz der Callback-Funktion in einem Aufruf der Funktion *SetWindowsHookEx* angeben.

CBTProc ist ein Platzhalter für den bibliotheksdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Bibliothek exportiert werden.

Die folgende Tabelle beschreibt die Parameter *wParam* und *lParam* für jede *HCBT_*-Konstante:

Konstante	Wirkung von *wParam* bzw. *lParam*
HCBT_ACTIVATE	*wParam:* Bestimmt das Handle des zu aktivierenden Fensters. *lparam:* Ein **long**-Zeiger auf eine *CBTACTIVATESTRUCT*-Struktur, die das Handle des aktuell aktiven Fensters enthält. Diese Struktur gibt außerdem an, ob sich die Aktivierung als Folge eines Mausklicks ändern soll.
HCBT_CLICKSKIPPED	*wParam:* Bezeichnet die Mausbotschaft, die aus der Botschaftswarteschlange des Systems entfernt wurde. *lparam:* Ein **long**-Zeiger auf eine *MOUSEHOOKSTRUCT*-Struktur, die den Treffertest-Code und das Handle des Fensters enthält, an das die Botschaft gerichtet ist. Eine Liste der Treffertest-Codes finden Sie in der Beschreibung der Botschaft *WM_NCHITTEST*.

Konstante	Wirkung von *wParam* bzw. *lParam*
HCBT_CREATE-WND	*wParam:* Bestimmt das Handle des neuen Fensters.
	lparam: Ein **long**-Zeiger auf eine *CBT_-CREATEWND*-Struktur, die Initialisierungsdaten für das Fenster enthält.
HBCT_-DESTROY-WND	*wParam:* Das Handle des freizugebenden Fensters.
	lparam: Dieser Parameter ist undefiniert und sollte auf 0L gesetzt werden.
HCBT_KEY-SKIPPED	*wParam:* Der virtuelle Tastencode.
	lparam: Bestimmt die Wiederholzahl, den Scancode den Code für die Tasten-Statusänderung, den vorherigen Tastenstatus und den Kontextcode. Weitere Informationen finden Sie in den Beschreibungen der Botschaften *WM_KEYUP* und *WM_KEY-DOWN*.
HCBT_-MINMAX	*wParam:* Das Handle des Fensters, das zum Vollbild vergrößert oder zum Symbol verkleinert werden soll.
	lparam: Das niederwertige *WORD* enthält einen *SW_-Wert*, der die Anzeigeoperation für das Fenster bestimmt. Eine Liste der möglichen Werte finden Sie in der Beschreibung der Funktion *ShowWindow*. Das höherwertige *WORD* ist nicht definiert.
HCBT_-MOVESIZE	*wParam:* Das Handle des Fensters, das verschoben oder in seiner Größe verändert werden soll.
	lparam: Ein **long**-Zeiger auf eine *RECT*-Struktur, die die Koordinaten des Fensters enthält.
HCBT_QS	*wParam:* Dieser Parameter ist nicht definiert und sollte auf 0 gesetzt werden.
	lparam: Dieser Parameter ist nicht definiert und sollte auf 0L gesetzt werden.
HCBT_-SETFOCUS	*wParam:* Das Handle des Fensters, das den Fokus erhält.
	lparam: Das niederwertige *WORD* enthält das Handle des Fensters, das den Fokus verliert. Das höherwertige *WORD* ist nicht definiert.

Konstante	Wirkung von *wParam* bzw. *lParam*
HCBT_SYS-COMMAND	*wParam:* Ein *SC_-Wert*, der einen Systembefehl bestimmt. Weitere Informationen über diese Werte finden Sie in der Beschreibung der Botschaft *WM_SYSCOMMAND*. *lparam:* Wenn *wParam* den Wert *SC_HOTKEY* hat, enthält das niederwertige *WORD* von *lParam* das Handle des Fensters, das durch eine Task-Umschaltung in den Vordergrund gestellt wird. Ist *wParam* ungleich *SC_HOTKEY* und wird ein Systembefehl mit der Maus ausgewählt, enthält das niederwertige *WORD* von *lParam* die x-Koordinate des Mauszeigers. Das höherwertige *WORD* enthält in diesem Fall die y-Koordinate des Mauszeigers. Trifft keine dieser Bedingungen zu, ist *lParam* undefiniert.

Siehe auch *CallNextHookEx, SetWindowsHookEx*

ChooseColor 3.1

Syntax

```
#include <commdlg.h>
BOOL ChooseColor(lpcc)

CHOOSECOLOR FAR* lpcc   /* Adresse der Struktur mit
                           den Initialisierungsdaten  */
```

Beschreibung Die Funktion *ChooseColor* erzeugt ein systemdefiniertes Dialogfenster, in dem der Anwender eine Farbe auswählen kann.

Parameter *lpcc:* Zeigt auf eine *CHOOSECOLOR*-Struktur, die die nötigen Informationen enthält, um das Dialogfenster zu initialisieren. Nach Rückkehr der Funktion *ChooseColor* enthält diese Struktur Informationen über die Farbwahl des Anwenders. Die *CHOOSECOLOR*-Struktur hat folgende Form:

```
#include <commdlg.h>

typedef struct tagCHOOSECOLOR {    /* cc */
    DWORD lStructSize;
    HWND hwndOwner;
    HWND hInstance;
    COLORREF rgbResult;
    COLORREF FAR* lpCustColors;
    DWORD Flags;
    LPARAM lCustData;
    UINT (CALLBACK* lpfnHook)(HWND, UINT,
```

```
            WPARAM, LPARAM);
    LPCSTR lpTemplateName;
} CHOOSECOLOR;
```

Rückgabewert

Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich ist. Er ist Null, wenn ein Fehler auftritt, der Anwender den Dialog abbricht oder mit Hilfe des Schließen-Befehls aus dem Systemmenü schließt.

Fehler

Verwenden Sie die Funktion *CommDlgExtendedError*, um den Fehlerwert zu ermitteln. Folgende Werte sind möglich:

CDERR_FINDRESFAILURE
CDERR_INITIALIZATION
CDERR_LOCKRESFAILURE
CDERR_LOADRESFAILURE
CDERR_LOADSTRFAILURE
CDERR_MEMALLOCFAILURE
CDERR_MEMLOCKFAILURE
CDERR_NOHINSTANCE
CDERR_NOHOOK
CDERR_NOTEMPLATE
CDERR_STRUCTSIZE

Hinweis

Das Dialogfenster unterstützt keine Farbpaletten. Die Farbauswahl, die das Dialogfenster zur Verfügung stellt, ist auf die Systemfarben beschränkt und enthält angenäherte Versionen dieser Farben.

Wenn die Filterfunktion – bezeichnet durch das Element *lpfnHook* der *CHOOSECOLOR*-Struktur – die *WM_CTLCOLOR*-Botschaft bearbeitet, muß diese Funktion das Handle des Pinsels zurückgeben, der zum Zeichnen des Hintergrunds des Steuerelements benutzt werden soll.

Beispiel

Das folgende Beispiel initialisiert eine *CHOOSECOLOR*-Struktur und erzeugt anschließend ein Dialogfenster zur Farbauswahl:

```
/* Variablen für die Farben */

CHOOSECOLOR cc;
COLORREF clr;
COLORREF aclrCust[16];
int i;

/* Alle Steuerelemente zum Einstellen der Farben
   werden mit Weiß initialisiert. */

for (i = 0; i < 16; i++)
    aclrCust[i] = RGB(255, 255, 255);

/* clr wird mit Schwarz initialisiert. */

clr = RGB(0, 0, 0);
```

```
/* Allen Feldern der Struktur wird Null zugewiesen. */

memset(&cc, 0, sizeof(CHOOSECOLOR));

/* Die benötigten Elemente von CHOOSECOLOR werden in
   geeigneter Weise initialisiert. */

cc.lStructSize = sizeof(CHOOSECOLOR);
cc.hwndOwner = hwnd;
cc.rgbResult = clr;
cc.lpCustColors = aclrCust;
cc.Flags = CC_PREVENTFULLOPEN;

if (ChooseColor(&cc))
    .
    . /* Verwenden Sie hier cc.rgbResult zur Auswahl
    .    der vom Anwender ausgewählten Farbe. */
    .
```

ChooseFont 3.1

Syntax

```
#include <commdlg.h>
BOOL ChooseFont(lpcf)

CHOOSEFONT FAR* lpcf;   /* Adresse der Struktur mit den
                           Initialisierungsdaten     */
```

Beschreibung

Die Funktion *ChooseFont* erzeugt ein systemdefiniertes Dialogfenster, in dem der Anwender eine Schrift, eine Schriftart, eine Punktgröße und eine Farbe auswählen kann.

Parameter

lpcf: Zeigt auf eine *CHOOSEFONT*-Struktur, die die benötigten Informationen enthält, um das Dialogfenster zu initialisieren. Nach Rückkehr der Funktion *ChooseFont* enthält diese Struktur Informationen über die Auswahl des Anwenders.

Die *CHOOSEFONT*-Struktur hat folgende Form:

```
#include <commdlg.h>

typedef struct tagCHOOSEFONT {   /* cf  */
    DWORD lStructSize;
    HWND hwndOwner;
    HDC hDC;
    LOGFONT FAR* lpLogFont;
    int iPointSize;
    DWORD Flags;
    COLORREF rbgColors;
    LPARAM lCustData;
    UINT (CALLBACK* lpfnHook)(HWND, UINT,
          WPARAM, LPARAM);
    LPCSTR lpTemplateName;
    HINSTANCE hInstance;
```

```
    LPSTR lpszStyle;
    UINT nFontType;
    int nSizeMin;
    int nSizeMax;
} CHOOSEFONT;
```

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich ausgeführt wurde. Er ist Null, wenn ein Fehler aufgetreten ist.

Fehler Die möglichen Fehlerwerte können Sie mit Hilfe der Funktion *CommDlgExtendedError* abrufen:

CDERR_FINDRESFAILURE
CDERR_INITIALIZATION
CDERR_LOCKRESFAILURE
CDERR_LOADRESFAILURE
CDERR_LOADSTRFAILURE
CDERR_MEMALLOCFAILURE
CDERR_MEMLOCKFAILURE
CDERR_NOHINSTANCE
CDERR_NOHOOK
CDERR_NOTEMPLATE
CDERR_STRUCTSIZE
CFERR_MAXLESSTHANMIN
CFERR_NOFONTS

Beispiel Das folgende Beispiel initialisiert eine *CHOOSEFONT*-Struktur und stellt danach das Dialogfenster zur Schriftauswahl dar:

```
LOGFONT lf;
CHOOSEFONT cf;

/* Allen Feldern der Struktur wird Null zugewiesen.   */

memset(&cf, 0, sizeof(CHOOSEFONT));

cf.lStructSize = sizeof(CHOOSEFONT);
cf.hwndOwner = hwnd;
cf.lpLogFont = &lf;
cf.Flags = CF_SCREENFONTS | CF_EFFECTS;
cf.rgbColors = RGB(0, 255, 255); /* hellblau */
cf.nFontType = SCREEN_FONTTYPE;

ChooseFont(&cf);
```

ClassFirst 3.1

Syntax

```
#include <toolhelp.h>
BOOL ClassFirst(lpce)

CLASSENTRY FAR* lpce;   /* Adresse der Struktur für
                           die Klasseninformationen   */
```

Beschreibung Die Funktion *ClassFirst* füllt die angegebene Struktur mit allgemeinen Informationen über die erste Klasse in der Klassenliste von Windows.

Parameter *lpce:* Zeigt auf eine *CLASSENTRY*-Struktur, die die Klasseninformationen aufnimmt. Diese Struktur hat folgende Form:

```
#include <toolhelp.h>
typedef struct tagCLASSENTRY {  /* ce */
   DWORD dwSize;
   HMODULE hInst;
   char szClassName[MAX_CLASSNAME + 1];
   WORD wNext;
} CLASSENTRY;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *ClassFirst* kann benutzt werden, um sich durch die Klassenliste von Windows zu bewegen. Um nachfolgende Elemente in der Klassenliste zu untersuchen, können Anwendungen die Funktion *ClassNext* benutzen.

Bevor *ClassFirst* aufgerufen wird, müssen Anwendungen die *CLASSENTRY*-Struktur initialisieren und deren Größe in Byte im Element *dwSize* angeben.

Detaillierte Informationen über eine bestimmte Klasse erhalten Sie mit Hilfe der Funktion *GetClassInfo*, indem Sie den Namen der Klasse und das Instanzen-Handle aus der *CLASSENTRY*-Struktur angeben.

Siehe auch *ClassNext*, *GetClassInfo*

ClassNext 3.1

Syntax

```
#include <toolhelp.h>
BOOL ClassNext(lpce)

CLASSENTRY FAR* lpce;   /* Adresse der Struktur für
                           die Klasseninformationen   */
```

Beschreibung Die Funktion *ClassNext* füllt die angegebene Struktur mit allgemeinen Informationen über die nächste Klasse in der Klassenliste von Windows.

Parameter *lpce:* Zeigt auf eine *CLASSENTRY*-Struktur, die die Klasseninformationen aufnimmt. Die *CLASSENTRY*-Struktur hat folgende Form:

```
#include <toolhelp.h>
typedef struct tagCLASSENTRY {   /* ce */
   DWORD dwSize;
   HMODULE hInst;
```

```
    char szClassName[MAX_CLASSNAME + 1];
    WORD wNext;
} CLASSENTRY;
```

Rückgabewert Bei erfolgreicher Ausführung der Funktion ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *ClassNext* kann benutzt werden, um sich weiter durch die Klassenliste von Windows zu bewegen, nachdem die Funktion *ClassFirst* aufgerufen wurde.

Detaillierte Informationen über eine bestimmte Klasse erhalten Sie mit Hilfe der Funktion *GetClassInfo*, indem Sie den Namen der Klasse und das Instanzen-Handle aus der *CLASSENTRY*-Struktur angeben.

Siehe auch *ClassFirst*

CloseDriver 3.1

Syntax

```
LRESULT CloseDriver(hdrvr, lParam1, lParam2)

HDRVR hdrvr;          /* Handle des installierbaren
                         Treibers                    */
LPARAM, lParam1;      /* treiberspezifische Daten    */
LPARAM, lParam2;      /* treiberspezifische Daten    */
```

Beschreibung Die Funktion *CloseDriver* schließt einen installierbaren Treiber.

Parameter *hdrvr:* Bezeichnet den installierbaren Treiber, der geschlossen werden soll. Der Wert für diesen Parameter muß aus einem vorhergehenden Aufruf der Funktion *OpenDriver* hervorgehen.

lParam1: Bestimmt treiberspezifische Daten.

lParam2: Bestimmt treiberspezifische Daten.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Ist der durch *hdrvr* bezeichnete Treiber beim Aufruf von *CloseDriver* die letzte Instanz dieses Treibers, ruft Windows die Funktion *DriverProc* dreimal auf. Beim ersten Aufruf setzt Windows den dritten Parameter von *DriverProc*, *wMessage*, auf *DRV_CLOSE*. Beim zweiten Aufruf setzt Windows *wMessage* auf *DRV_DISABLE*. Beim dritten Aufruf setzt Windows *wMessage* auf *DRV_FREE*. Wenn der durch *hdrvr* bezeichnete Treiber nicht die letzte Instanz dieses Treibers ist, wird die Funktion *DriverProc* nur mit dem Wert *DRV_CLOSE* für *wMessage* aufgerufen. Die in den Parametern *lParam1* und *lParam2* angegebenen Werte werden in den Parametern *lParam1* und *lParam2* der Funktion *DriverProc* übergeben.

Siehe auch *DriverProc*, *OpenDriver*

CommDlgExtendedError 3.1

Syntax

```
#include <commdlg.h>
DWORD CommDlgExtendedError(VOID);
```

Beschreibung Tritt bei der Ausführung einer der folgenden Standarddialogfensterfunktionen aus der dynamischen Linkbibliothek COMMDLG.DLL ein Fehler auf, kann dessen Grund mit der Funktion *CommDlgExtendedError* ermittelt werden:

ChooseColor
ChooseFont
FindText
GetFileTitle
GetOpenFileName
GetSaveFileName
PrintDlg
ReplaceText

Parameter Diese Funktion hat keine Parameter.

Rückgabewert Wenn der vorhergehende Aufruf einer solchen Dialogfensterfunktion erfolgreich war, ist der Rückgabewert Null. Er ist *CDERR_DIALOGFAILURE*, wenn das Dialogfenster nicht erzeugt werden konnte. In allen anderen Fällen ist der Rückgabewert ein Integer ungleich Null, der eine Fehlerbedingung bezeichnet.

Hinweis Die folgende Liste beschreibt die möglichen Rückgabewerte der Funktion *CommDlgExtendedError* und deren Bedeutung:

Wert	Bedeutung
CDERR_FINDRES-FAILURE	Die Dialogfensterfunktion konnte eine angegebene Ressource nicht finden.
CDERR_INITIA-LIZATION	Die Dialogfensterfunktion ist während der Initialisierung fehlgeschlagen. Dieser Fehler tritt meist dann auf, wenn nicht genügend Speicher vorhanden ist.
CDERR_LOADRES-FAILURE	Die Dialogfensterfunktion konnte die angegebene Ressource nicht laden.
CDERR_LOCKRES-FAILURE	Die Dialogfensterfunktion konnte eine angegebene Ressource nicht sperren.
CDERR_LOADSTR-FAILURE	Die Dialogfensterfunktion konnte den angegebenen String nicht laden.
CDERR_MEM-ALLOCFAILURE	Die Dialogfensterfunktion konnte keinen Speicherplatz für interne Datenstrukturen reservieren.

Wert	Bedeutung
CDERR_MEMLOCK-FAILURE	Die Dialogfensterfunktion konnte den Speicherplatz, der einem Handle zugeordnet ist, nicht sperren.
CDERR_NOH-INSTANCE	Das *ENABLETEMPLATE*-Flag im Element *Flags* einer Datenstruktur für Dialogfenster wurde zwar gesetzt, die Anwendung konnte aber kein entsprechendes Instanzen-Handle bereitstellen.
CDERR_NOHOOK	Das *ENABLEHOOK*-Flag im Element *Flags* einer Datenstruktur für Dialogfenster wurde zwar gesetzt, die Anwendung konnte aber keinen Zeiger auf eine entsprechende Filterfunktion bereitstellen.
CDERR_NO-TEMPLATE	Das *ENABLETEMPLATE*-Flag im Element *Flags* einer Datenstruktur für Dialogfenster wurde zwar gesetzt, die Anwendung konnte aber keine entsprechende Schablone bereitstellen.
CDERR_REGISTER-MSGFAIL	Die Funktion *RegisterWindowMessage* hat nach Aufruf durch die Dialogfensterfunktion einen Fehlerwert zurückgegeben.
CDERR_STRUCT-SIZE	Das Element *lStructSize* der Datenstruktur für ein Dialogfenster ist ungültig.
CFERR_NOFONTS	Es existieren keine Schriften.
CFERR_MAXLESS-THANMIN	Die angegebene maximale Größe des Dialogfensters ist kleiner als die angegebene minimale Größe.
FNERR_BUFFER-TOOSMALL	Der Puffer für den Dateinamen ist zu klein. Das Element *lpstrFile* einer Datenstruktur für Standarddialogfenster zeigt auf diesen Puffer.
FNERR_INVALID-FILENAME	Der angegebene Dateiname ist ungültig.
FNERR_SUBCLASS-FAILURE	Der Versuch der Unterklassifizierung einer Liste ist wegen Speichermangels fehlgeschlagen.
FRERR_BUFFER-LENGTHZERO	Ein Element einer Datenstruktur für Standarddialogfenster zeigt auf einen ungültigen Puffer.

Wert	Bedeutung
PDERR_CREATE-ICFAILURE	Die Funktion *PrintDlg* konnte keinen Informationskontext erzeugen.
PDERR_DEFAULT-DIFFERENT	Eine Anwendung hat die Funktion *PrintDlg* mit gesetztem Flag *DN_DEFAULTPRN* im Element *wDefault* der *DEVNAMES*-Struktur aufgerufen. Der durch andere Strukturelemente beschriebene Drucker entspricht jedoch nicht dem Standarddrucker. Dieser Fehler tritt auf, wenn eine Anwendung die *DEVNAMES*-Struktur speichert und der Anwender den Standarddrucker in der Systemsteuerung wechselt. Um den in der *DEVNAMES*-Struktur beschriebenen Drucker zu verwenden, sollte die Anwendung das Flag *DN_DEFAULTPRN* zurücksetzen und die Funktion *PrintDlg* erneut aufrufen. Wenn der Standarddrucker verwendet werden soll, sollte die Anwendung die *DEVNAMES*-Struktur zurücksetzen. Wird gleichzeitig die *DEVMODE*-Struktur durch *NULL* ersetzt, wird der Standarddrucker automatisch ausgewählt.
PDERR_DNDM-MISMATCH	Die Daten in den *DEVMODE*- und *DEVNAMES*-Datenstrukturen beschreiben zwei verschiedene Drucker.
PDERR_GETDEV-MODEFAIL	Der Druckertreiber konnte keine *DEVMODE*-Datenstruktur initialisieren. Dieser Fehlerwert tritt nur bei Druckertreibern auf, die für Windows 3.0 oder spätere Versionen geschrieben wurden.
PDERR_INIT-FAILURE	Die Funktion *PrintDlg* schlug während der Initialisierung fehl.
PDERR_LOADDRV-FAILURE	Die Funktion *PrintDlg* konnte den Gerätetreiber für den angegebenen Drucker nicht laden.
PDERR_NODE-FAULTPRN	Es existiert kein voreingestellter Standarddrucker.
PDERR_NODEVICES	Es wurden keine Druckertreiber gefunden.
PDERR_PARSE-FAILURE	Die Funktion *PrintDlg* konnte die Strings im Abschnitt [devices] der Datei WIN.INI nicht analysieren.

Wert	Bedeutung
PDERR_PRINTER-NOTFOUND	Der Abschnitt [devices] in der Datei WIN.INI enthält keinen Eintrag für den angeforderten Drucker.
PDERR_RETDEF-FAILURE	Das *PD_RETURNDEFAULT*-Flag im Element *Flags* der *PRINTDLG*-Datenstruktur wurde zwar gesetzt, aber entweder war das Feld *hDevMode* oder *hDevNames* ungleich Null.
PDERR_SETUP-FAILURE	Die Funktion *PrintDlg* konnte die benötigten Ressourcen nicht laden.

Siehe auch *ChooseColor, ChooseFont, FindText, GetFileTitle, GetOpenFile-Name, GetSaveFileName, PrintDlg, ReplaceText*

CopyCursor 3.1

Syntax

```
HCURSOR CopyCursor(hinst, hcur)

HINSTANCE hinst;      /* Handle der Anwendungsinstanz */
HCURSOR hcur;         /* Handle des zu kopierenden
                         Mauszeigers                  */
```

Beschreibung Die Funktion *CopyCursor* kopiert einen Mauszeiger.

Parameter *hinst:* Bezeichnet die Instanz des Moduls, die den Mauszeiger kopiert.

hcur: Bezeichnet den zu kopierenden Mauszeiger.

Rückgabewert Der Rückgabewert ist das Handle der Kopie des Mauszeigers, wenn die Funktion erfolgreich ist. Andernfalls ist der Rückgabewert *NULL*.

Hinweis Anwendungen müssen den Mauszeiger mit Hilfe der Funktion *DestroyCursor* freigeben, wenn dieser nicht länger benötigt wird.

Die Funktion *CopyCursor* ermöglicht es Anwendungen oder dynamischen Linkbibliotheken, die Mauszeiger anderer Module zu benutzen. Da alle Ressourcen dem Modul zugeordnet sind, aus dem sie stammen, kann eine Ressource nicht gemeinsam genutzt werden, nachdem das Modul freigegeben wurde.

CopyCursor ermöglicht es einer Anwendung eine Kopie zu erstellen, die dann dieser Anwendung zugeordnet ist.

Siehe auch *CopyIcon, DestroyCursor, GetCursor, SetCursor, ShowCursor*

CopyIcon 3.1

Syntax

```
HICON CopyIcon(hinst, hicon)

HINSTANCE hinst; /* Handle der Anwendungsinstanz */
HICON hicon;     /* Handle des zu kopierenden Symbols */
```

Beschreibung Die Funktion *CopyIcon* kopiert ein Symbol.

Parameter *hinst:* Bezeichnet die Instanz des Moduls, die das Symbol kopiert.

hicon: Bezeichnet das zu kopierende Symbol.

Rückgabewert Wenn die Funktion erfolgreich war, ist der Rückgabewert das Handle der Kopie des Symbols. Andernfalls ist er *NULL*.

Hinweis Wenn eine Anwendung ein Symbol nicht mehr benötigt, sollte sie es mit Hilfe der Funktion *DestroyIcon* freigeben.

Die Funktion *CopyIcon* ermöglicht es Anwendungen oder dynamischen Linkbibliotheken, Symbole anderer Module zu verwenden. Da alle Ressourcen dem Modul zugeordnet sind, aus dem sie stammen, kann eine Ressource nicht gemeinsam genutzt werden, nachdem das Modul freigegeben wurde. *CopyIcon* ermöglicht es einer Anwendung, eine Kopie eines Symbols zu erzeugen, die dann dieser Anwendung zugeordnet ist.

Siehe auch *CopyCursor, DestroyIcon, DrawIcon*

CopyLZFile 3.1

Syntax

```
#include <lzexpand.h>
LONG CopyLZFile(hSource, hfDest)

HFILE hfSource;      /* Handle der Quelldatei      */
HFILE hfDest;        /* Handle der Zieldatei       */
```

Beschreibung Die Funktion *CopyLZFile* kopiert eine Quelldatei in eine Zieldatei. Wenn die Quelldatei komprimiert ist, erstellt diese Funktion eine entkomprimierte Zieldatei. Ist die Quelldatei nicht komprimiert, wird die Ausgangsdatei nur kopiert.

Parameter *hfSource:* Bezeichnet die Quelldatei.

hfDest: Bezeichnet die Zieldatei.

Rückgabewert Der Rückgabewert bestimmt die Größe der Zieldatei in Byte, wenn die Funktion erfolgreich war, andernfalls ist er ein Fehlerwert kleiner Null. Möglich sind folgende Werte:

Wert	Bedeutung
LZERROR_BADINHANDLE	Das Handle, das die Quelldatei bezeichnet, war ungültig.

Wert	Bedeutung
LZERROR_-BADOUTHANDLE	Das Handle, das die Zieldatei bezeichnet, war ungültig.
LZERROR_READ	Das Format der Quelldatei war ungültig.
LZERROR_WRITE	Es steht nicht genügend Speicherplatz für die Ausgabedatei zur Verfügung.
LZERROR_GLOB-ALLOC	Es steht nicht genügend Speicher für die benötigten Puffer zur Verfügung.
LZERROR_-UNKNOWNALG	Die Datei wurde mit einem unbekannten Algorithmus komprimiert.

Hinweis

Die Funktion *CopyLZFile* dient zum Kopieren und/oder Entkomprimieren mehrerer Dateien. Anwendungen sollten die Funktion *LZStart* vor *CopyLZfile* aufrufen, um die benötigten Puffer zu reservieren. Nach dem Kopieren der Dateien sollten Anwendungen die Funktion *LZDone* aufrufen, um diese Puffer freizugeben.

Wenn die Funktion erfolgreich ist, liegt die durch *hfDest* bezeichnete Datei entkomprimiert vor.

Wenn die Quell- oder Zieldatei mit einer Standardbibliotheksfunktion von C anstelle von *_lopen* oder *OpenFile* geöffnet wird, muß sie im Binärmodus geöffnet werden.

Beispiel

Das folgende Beispiel ruft die Funktion *CopyLZFile* auf, um Kopien von vier Textdateien zu erzeugen:

```
#define STRICT

#include <windows.h>
#include <lzexpand.h>

#define NUM_FILES   4

char *szSrc[NUM_FILES] =
    {"readme.txt", "data.txt", "update.txt",
     "list.txt"};
char *szDest[NUM_FILES] =
    {"readme.bak", "data.bak", "update.bak",
     "list.bak"};
OFSTRUCT ofStrSrc;
OFSTRUCT ofStrDest;
HFILE hfSrcFile, hfDstFile;
int i;

/* Interne Puffer für die Funktion CopyLZFile
   reservieren. */

LZStart ();
```

```
/* Dateien öffnen, kopieren und schließen. */

for (i = 0; i < NUM_FILES; i++) {
   hfSrcFile = LZOpenFile(szSrc[i], &ofStrSrc,
               OF_READ);
   hfDstFile = LZOpenFile(szDest[i], &ofStrDest,
                          OF_CREATE);
   CopyLZFile(hfSrcFile, hfDstFile);
   LZClose(hfSrcFile);
   LZClose(hfDstFile);
}

LZDone(); /* Interne Puffer freigeben. */
```

Siehe auch _lopen, *LZCopy*, *LZDone*, *LZStart*, *OpenFile*

CPIApplet 3.1

Syntax

```
LONG CALLBACK* CPIApplet(hwndCPl, iMessage, lParam1,
   lParam2)

HWND hwndCPl;       /* Handle des Fensters der
                       Systemsteuerung               */
UINT iMessage;      /* Botschaft                     */
LPARAM lParam1;     /* Erster Botschaftsparameter    */
LPARAM lParam2      /* Zweiter Botschaftsparameter   */
```

Beschreibung Die Funktion *CPIApplet* dient als Einsprungspunkt für eine dynamische Linkbibliothek (DLL) der Systemsteuerung. Diese Funktion wird von der Anwendung bereitgestellt.

Parameter *hwndCPl:* Bezeichnet das Hauptfenster der Systemsteuerung.

iMessage: Bestimmt die Botschaft, die an die DLL übergeben wird.

lParam1: Bestimmt 32 Bits zusätzlicher, botschaftsabhängiger Informationen.

lParam2: Bestimmt 32 Bits zusätzlicher, botschaftsabhängiger Informationen.

Rückgabewert Der Rückgabewert hängt von der Botschaft ab.

Hinweis Verwenden Sie den Parameter *hwndCPl* für Dialogfenster oder andere Fenster, die ein Handle ihres übergeordneten Fensters benötigen.

CreateScalableFontResource 3.1

Syntax

```
BOOL CreateScalableFontResource(fHidden,
   lpszResourceFile, lpszFontFile, lpszCurrentPath)
```

```
UINT fHidden;            /* Flag für eingebettete
                            Nur-Lesen-Schrift       */
LPCSTR lpszResourceFile; /* Adresse des Dateinamens
                            der Schriftressource    */
LPCSTR lpszFontFile;     /* Adresse des Dateinamens
                            der skalierbaren Schrift */
LPCSTR lpszCurrentPath;  /* Adresse des Pfads der
                            Schriftdatei            */
```

Parameter

fHidden: Bestimmt, ob es sich um eine eingebettete Nur-Lesen-Schrift handelt:

Wert	Bedeutung
0	Für die Schrift existiert Lese- und Schreibzugriffsberechtigung.
1	Die Schrift kann nur gelesen werden und wird vor allen anderen Anwendungen im System verborgen. Wenn dieses Flag gesetzt ist, wird die Schrift von den Aufzählungsfunktionen *EnumFonts* und *EnumFontFamilies* nicht aufgelistet.

lpszResourceFile: Zeigt auf einen nullterminierten String, der den Namen der Schriftressourcendatei bestimmt, die von dieser Funktion erzeugt wird.

lpszFontFile: Zeigt auf einen nullterminierten String, der den Namen der Datei einer skalierbaren Schrift bestimmt. Diese Datei wird von *CreateScalableFontResource* verwendet, um die Schriftressourcendatei zu erzeugen. Dieser Parameter muß entweder den Dateinamen und die Namenserweiterung oder den vollständigen Pfad (Laufwerk, Pfad, Dateiname und Namenserweiterung) angeben.

lpszCurrentPath: Zeigt auf einen nullterminierten String, der den Pfad der Datei der skalierbaren Schrift bestimmt, die in *lpszFontFile* angegeben ist. Dieser Parameter ist *NULL*, wenn *lpszFontFile* bereits den vollständigen Pfadnamen enthält.

Rückgabewert

Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich war. Andernfalls ist der Rückgabewert Null.

Hinweis

Anwendungen müssen die Funktion *CreateScalableFontResource* zum Erzeugen einer Schriftressourcendatei aufrufen, bevor sie eine eingebettete Schrift installieren. Schriftressourcendateien für Schriften, die Lese- und Schreibzugriffsberechtigung besitzen, sollten die Dateinamenserweiterung .FOT verwenden. Schriftressourcendateien für Schriften, die nur die Lesezugriffsberechtigung besitzen, sollten eine andere Namenserweiterung (z.B. .FOR) verwenden. Außerdem sollten sie vor anderen Anwendungen verborgen werden, indem im Parameter *fHidden* der Wert 1 angegeben wird. Die Schriftressourcen-

dateien können mit der Funktion *AddFontResource* installiert werden.

Enthält der Parameter *lpszFontFile* nur den Dateinamen und die Namenserweiterung, muß der Parameter *lpszCurrentPath* den Pfad angeben. Bestimmt der Parameter *lpszFontFile* bereits einen vollständigen Pfad, muß der Parameter *lpszCurrentPath* *NULL* oder ein Zeiger auf *NULL* sein.

Sind Dateiname und Namenserweiterung im Parameter *lpszFontFile* und der Pfad im Parameter *lpszCurrentPath* angegeben, wird der String in *lpszFontFile* als Name der zur Ressource gehörende .TTF-Datei in die .FOT-Datei kopiert. Beim Aufruf der Funktion *AddFontResource* unterstellt das System, daß die .TTF-Datei in das Windows-Systemverzeichnis bzw. in das Windows-Hauptverzeichnis bei einer Netzinstallation kopiert wurde. Beim Aufruf der Funktion *CreateScalableFontResource* muß sich die .TTF-Datei nicht in diesem Verzeichnis befinden, da der Parameter *lpszCurrentPath* die Verzeichnisinformationen enthält. Eine so erzeugte Ressource enthält keine absoluten Pfadinformationen und kann deshalb in jeder Windows-Installation verwendet werden.

Auch wenn der Parameter *lpszFontFile* den Pfad und *lpszCurrentPath* *NULL* enthält, wird der String in *lpszFontFile* in die .FOT-Datei kopiert. In diesem Fall muß sich allerdings die .TTF-Datei beim Aufruf der Funktion *AddFontResource* an der Position befinden, die beim Aufruf der Funktion *CreateScalableFontResource* im Parameter *lpszFontFile* angegeben war. Der Parameter *lpszCurrentPath* wird dann nicht benötigt. Eine so erzeugte Ressource enthält absolute Angaben für Laufwerk und Pfad. Sie arbeitet deshalb nicht, wenn sich die .TTF-Datei nicht mehr an der angegebenen Position befindet.

Die Funktion *CreateScalableFontResource* unterstützt nur die skalierbaren TrueType-Schriften.

Beispiel

Das folgende Beispiel zeigt die Erstellung einer TrueType-Schriftdatei im Verzeichnis SYSTEM oder im Startverzeichnis von Windows:

```
CreateScalableFontResource(0,
   "c:\\windows\\system\\font.fot", "font.ttr",
   "c:\\windows\\system");

AddFontResource("c:\\windows\\system\\font.fot");
```

Das folgende Beispiel zeigt die Erstellung einer TrueType-Schriftdatei in einem bestimmten Verzeichnis:

```
CreateScalableFontResource(0,
   "c:\\windows\\system\\font.fot",
   "c:\\fondir\\font.ttr", NULL);

AddFontResource("c:\\windows\\system\\font.fot");
```

Das folgende Beispiel zeigt die Verwendung einer eingebetteten Standardschrift:

```
HFONT hfont;

/* .TTF_Datei in C:\MYDIR\FONT.TTR
   extrahieren. */

CreateScalableFontResource(0, "font.fot",
   "c:\\mydir\\font.ttr", NULL);

AddFontResource("font.fot");

hfont = CreateFont(..., CLIP_DEFAULT_PRECIS,
                   ..., "FONT");

    .
    . /* Die Schrift benutzen. */
    .
DeleteObject(hfont);

RemoveFontResource("font.fot");

    .
    . /* C:\MYDIR\FONT.FOT und
    .    C:\MYDIR\FONT.TTR löschen. */
    .
```

Das folgende Beispiel zeigt die Verwendung einer eingebetteten Nur-Lesen-Schrift:

```
HFONT hfont;

/* .TTF-Datei in C:\MYDIR\FONT.TTR
    extrahieren. */

CreateScalableFontResource(1, "font.for",
   "c:\\mydir\\font.ttr", NULL);

AddFontResource("font.for");

hfont = CreateFont(..., CLIP_EMBEDDED, ..., "FONT");
    .
    . /* Die Schrift benutzen. */
    .
DeleteObject(hfont);

RemoveFontResource("font.for");

    .
    . /* C:\MYDIR\FONT.FOR und C:\MYDIR\FONT.TTR
    .    löschen. */
    .
```

Siehe auch *AddFontResource*

DdeAbandonTransaction 3.1

Syntax

```
#include <ddeml.h>
BOOL DdeAbandonTransaction(idInst, hConv, idTransaction)

DWORD idInst;         /* Instanzenbezeichner         */
HCONV hConv;          /* Handle der Konversation     */
DWORD idTransaction;  /* Transaktionsbezeichner      */
```

Beschreibung Die Funktion *DdeAbandonTransaction* unterbricht die angegebene asynchrone Transaktion und gibt alle Ressourcen frei, die der Transaktion zugeordnet sind.

Parameter *idInst:* Gibt den Bezeichner der Anwendungsinstanz an, der durch einen früheren Aufruf der Funkton *DdeInitialize* ermittelt wurde.

hConv: Bezeichnet die Konversation, in der die Transaktion eingeleitet wurde. Wenn der Parameter *NULL* ist, werden alle Transaktionen abgebrochen und der Parameter *idTransaction* wird ignoriert.

idTransaction: Bezeichnet die Transaktion, die beendet werden soll. Wenn der Parameter *NULL* ist, werden alle aktiven Transaktionen in der angegebenen Konversation abgebrochen.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich war. Andernfalls ist er Null.

Fehler Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_DLL_NOT_INITIALIZED
DMLERR_INVALIDPARAMETER
DMLERR_NO_ERROR
DMLERR_UNFOUND_QUEUE_ID

Hinweis Die Funktion *DdeAbandonTransaction* sollte nur von einer DDE-Client-Anwendung aufgerufen werden. Wenn die Server-Anwendung auf die Transaktion antwortet, nachdem der Client die Funktion *DdeAbandonTransaction* aufgerufen hat, verwirft das System die Ergebnisse der Transaktion.

Diese Funktion hat keine Auswirkung auf synchrone Transaktionen.

Siehe auch *DdeClientTransaction, DdeGetLastError, DdeInitialize, DdeQueryConvInfo*

DdeAccessData 3.1

Syntax

```
#include <ddeml.h>
BYTE FAR* DdeAccessData(hData, lpcbData)
```

```
HDDEDATA hData;        /* Handle eines globalen
                          Speicherobjekts            */
DWORD FAR* lpcbData;   /* Zeiger auf die Variable,
                          die die Datenlänge erhält  */
```

Beschreibung Die Funktion *DdeAcessData* ermöglicht den Zugriff auf die Daten im angegebenen globalen Speicherobjekt. Wenn eine Anwendung den Datenzugriff im Objekt abgeschlossen hat, muß sie die Funktion *DdeUnaccessData* aufrufen.

Parameter *hData:* Bezeichnet das globale Speicherobjekt, auf das zugegriffen werden soll.

lpcbData: Zeigt auf eine Variable, die die Größe des durch den Parameter *hData* angegebenen globalen Speicherobjekts in Byte aufnimmt. Wenn dieser Parameter *NULL* ist, wird keine Information über die Größe zurückgeliefert.

Rückgabewert Wenn die Funktion erfolgreich war, zeigt der Rückgabewert auf das erste Byte im globalen Speicherobjekt. Andernfalls ist der Rückgabewert *NULL*.

Fehler Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_DLL_NOT_INITIALIZED
DMLERR_INVALIDPARAMETER
DMLERR_NO_ERROR

Hinweis Wenn der Parameter *hData* nicht an eine Funktion der DDE-Bibliothek (DDEML) übergeben wurde, kann eine Anwendung den Zeiger, der von der Funktion *DdeAccessData* zurückgeliefert wurde, für Schreib- und Lesezugriff auf das globale Speicherobjekt verwenden. Wenn *hData* jedoch bereits an eine DDEML-Funktion übergeben wurde, kann der Zeiger nur für Lesezugriffe auf das Speicherobjekt verwendet werden.

Beispiel Das folgende Beispiel ruft die Funktion *DdeAccessData* auf, um einen Zeiger auf ein globales Speicherobjekt zu ermitteln. Der Zeiger dient dann zum Kopieren der Daten aus dem Objekt in einen lokalen Puffer. Anschließend wird der Zeiger freigegeben:

```
HDDEDATA hData;
LPBYTE lpszAdviseData;
DWORD cbDataLen;
DWORD i;
char szData[128];

lpszAdviseData = DdeAccessData(hData, &cbDataLen);
for (i = 1; i < cbDataLen; i++)
    szData[i] = *lpszAdviseData++;
DdeUnaccessData(hData);
```

Siehe auch *DdeAddData, DdeCreateDataHandle, DdeFreeDataHandle, DdeGetLastError, DdeUnaccessData*

DdeAddData 3.1

Syntax

```
#include <ddeml.h>
HDDEDATA DdeAddData(hData, lpvSrcBuf, cbAddData, offObj)

HDDEDATA hData;       /* Handle des globalen
                         Speicherobjekts             */
void FAR* lpvSrcBuf;  /* Adresse des Quellpuffers    */
DWORD cbAddData;      /* Länge der Daten             */
DWORD offObj;         /* Offset innerhalb des globalen
                         Speicherobjekts             */
```

Beschreibung Die Funktion *DdeAddData* fügt dem angegebenen globalen Speicherobjekt Daten hinzu. Eine Anwendung kann Daten an einem beliebigen Offset ab dem Anfang des Speicherobjekts anfügen. Wenn die neuen Daten die bereits im Objekt gespeicherten Daten überlappen, werden die alten Daten durch die neuen in den Byte überschrieben, in denen die Überlappung auftritt. Die Inhalte der Positionen im Objekt, an die nicht geschrieben wurde, sind undefiniert.

Parameter *hData:* Bezeichnet das globale Speicherobjekt, das die zusätzlichen Daten erhält.

lpvSrcBuf: Zeigt auf einen Puffer, der die Daten enthält, die dem globalen Speicherobjekt hinzugefügt werden sollen.

cbAddData: Gibt die Länge der Daten in Byte an, die dem globalen Speicherobjekt hinzugefügt werden sollen.

offObj: Gibt den Offset vom Anfang des globalen Speicherobjekts in Byte an. Die zusätzlichen Daten werden ab diesem Offset in das Objekt kopiert.

Rückgabewert Wenn die Funktion erfolgreich war, ist der Rückgabewert das neue Handle des globalen Speicherobjekts. Dieses neue Handle sollte bei allen Verweisen auf das Objekt benutzt werden. Der Rückgabewert ist Null, wenn ein Fehler aufgetreten ist.

Fehler Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_DLL_NOT_INITIALIZED
DMLERR_INVALIDPARAMETER
DMLERR_MEMORY_ERROR
DMLERR_NO_ERROR

Hinweis Nachdem ein Daten-Handle als Parameter einer anderen Funktion der DDE-Bibliothek (DDEML) benutzt oder von einer DDE-Callback-Funktion zurückgegeben wurde, darf das Hand-

le nur für Lesezugriffe auf das bezeichnete globale Speicherobjekt verwendet werden.

Wenn die Größe des globalen Speicherst nicht ausreicht, um die hinzuzufügenden Daten aufzunehmen, reserviert die Funktion *DdeAddData* das globales Speicherobjekt in der benötigten Größe.

Beispiel

Das folgende Beispiel erzeugt ein globales Speicherobjekt. Mit der Funktion *DdeAddData* werden dem Objekt Daten hinzugefügt, die dann mit der Transaktion *XTYP_POKE* an den Client übergeben werden:

```
DWORD IDInst;         /* Instanzenbezeichner */
HDDEDATA hddeStrings; /* Daten-Handle */
HSZ hszMyItem;        /* String-Handle des
                         Elementnamens */
DWORD offObj = 0;     /* Offset im globalen
                         Speicherobjekt */
char szMyBuf[16];     /* Puffer für temporären
                         String */
HCONV hconv;          /* Handle der Konversation */
DWORD dwResult;       /* Ergebnisse der Transaktion */
BOOL fAddAString;     /* TRUE, wenn hinzuzufügende
                         Strings existieren */

/* Globales Speicherobjekt erzeugen. */

hddeStrings = DdeCreateDataHandle(idInst, NULL, 0, 0,
    hszMyItem, CF_TEXT, 0);

/* Wenn ein String-Handle verfügbar ist, wird er von
 * der anwendungsdefinierten Funktion IsThereAString()
 * in szMyBuf kopiert. Die Funktion gibt in diesem Fall
 * TRUE, andernfalls FALSE zurück.
 */

while ((fAddAString = IsThereAString())) {

/* Den String zum globalen Speicherobjekt hinzufügen. */

   DdeAddData(hddeStrings,          /* Daten-Handle */
       &szMyBuf,                    /* String-Puffer */
       (DWORD) strlen(szMyBuf) + 1, /* Zeichenzähler */
       offObj;                      /* Offset im
                                       Objekt */

   offObj = (DWORD) strlen(szMyBuf) + 1;
                                 /* Offset ausrichten */

}

/* Daten werden dem Server übergeben, da keine
 * hinzuzufügenden Daten mehr existieren.
 */
```

```
DdeClientTransaction((void FAR*) hddeStrings, -1L,
    hconv, hszMyItem, CF_TEXT, XTYP_POKE, 1000,
    &dwResult);
```

Siehe auch *DdeAccessData, DdeCreateHandle, DdeGetLastError, DdeUnaccessData*

DdeCallback 3.1

Syntax

```
#include <ddeml.h>
HDDEDATA CALLBACK DdeCallback(type, fmt,hconv, hsz1,
   hsz2, hData, dwData1, dwData2)

UNIT type;        /* Typ der Transaktion             */
UNIT fmt;         /* Datenformat der Zwischenablage  */
HCONV hconv;      /* Handle der Konversation         */
HSZ hsz1;         /* Handle eines String             */
HSZ hsz2;         /* Handle eines String             */
HDDEDATA hData;   /* Handle eines globalen
                     Speicherobjekts                 */
DWORD dwData1;    /* transaktionsspezifische Daten   */
DWORD dwData2;    /* transaktionsspezifische Daten   */
```

Beschreibung Die Funktion *DdeCallback* ist eine anwendungsdefinierte DDE-Callback-Funktion, die DDE-Transaktionen bearbeitet. Diese DDE-Transaktionen werden als Ergebnis des Aufrufs anderer Anwendungen von der DDEML an die Funktion gesendet.

Parameter *type:* Bezeichnet den Typ der aktuellen Transaktion. Dieser Parameter besteht aus einer Kombination von Transaktionsklassen-Flags und Transaktionstyp-Flags. Die folgende Tabelle beschreibt jede der Transaktionsklassen und liefert eine Liste der Transaktionstypen in jeder Klasse:

Wert	Bedeutung
XCLASS_BOOL	Eine DDE-Callback-Funktion sollte, nachdem sie eine Transaktion bearbeitet hat, die zu ihrer Klasse gehört, *TRUE* oder *FALSE* zurückliefern. Im folgenden sind die *XCLASS_BOOL*-Transaktionstypen aufgeführt: *XTYP_ADVSTART, XTYP_CONNECT*
XCLASS_DATA	Eine DDE-Callback-Funktion sollte, nachdem sie eine Transaktion bearbeitet hat, die zu ihrer Klasse gehört, ein DDE-Daten-Handle, *CBR_BLOCK* oder *NULL* zurückliefern. Im folgenden sind die *XCLASS_DATA*-Transaktionstypen aufgeführt: *XTYP_ADVREQ, XTYP_REQUEST, XTYP_WILDCONNECT*

Wert	Bedeutung
XCLASS_FLAGS	Eine DDE-Callback-Funktion sollte, nachdem sie eine Transaktion bearbeitet hat, die zu ihrer Klasse gehört, *DDE_FACK*, *DDE_FBUSY* oder *DDE_FNOTPROCESSED* zurückliefern. Im folgenden sind die *XCLASS_FLAGS*-Transaktionstypen aufgeführt: *XTYP_ADVDATA, XTYP_EXECUTE, XTYP_POKE*
XCLASS_-NOTIFICATION	Die Transaktionstypen, die zu dieser Klasse gehören, dienen nur Benachrichtigungszwekken. Der Rückgabewert der Callback-Funktion wird ignoriert. Im folgenden sind die *XCLASS_NOTIFICATION*-Transaktionstypen aufgeführt: *XTYP_ADVSTOP, XTYP_CONNECT_CONFIRM, XTYP_DISCONNECT, XTYP_ERROR, XTYP_MONITOR, XTYP_REGISTER, XTYP_XACT_COMPLETE, X,TYP_UNREGISTER*

fmt: Bezeichnet das Format, in dem die Daten gesendet oder empfangen werden.

hconv: Bezeichnet die Konversation, die der aktuellen Transaktion zugeordnet ist.

hsz1: Bezeichnet einen String. Die Bedeutung dieses Parameters hängt vom Typ der aktuellen Transaktion ab. In der Beschreibung der Transaktionstypen finden Sie weitere Informationen.

hsz2: Bezeichnet einen String. Die Bedeutung dieses Parameters hängt vom Typ der aktuellen Transaktion ab. In der Beschreibung der Transaktionstypen finden Sie weitere Informationen.

hData: Bezeichnet DDE-Daten. Die Bedeutung dieses Parametes hängt vom Typ der aktuellen Transaktion ab. In der Beschreibung der Transaktionstypen finden Sie weitere Informationen.

dwData1: Bezeichnet transaktionsspezifische Daten. Weitere Informationen finden Sie in der Beschreibung der Transaktionstypen.

dwData2: Bezeichnet transaktionsspezifische Daten. Weitere Informationen finden Sie in der Beschreibung der Transaktionstypen.

Rückgabewert Der Rückgabewert hängt von der Transaktionsklasse ab.

Hinweis

Die Callback-Funktion wird asynchron für Transaktionen aufgerufen, die die Erzeugung oder Beendigung von Konversationen nicht beeinflussen. Eine Anwendung, die eingehende Botschaften nur selten bearbeitet, reduziert die DDE-Ausführungsgeschwindigkeit, da die DDEML Botschaften zur Einleitung von Transaktionen benutzt.

Eine Anwendung muß eine Callback-Funktion registrieren, indem sie ihre Adresse in einem Aufruf der Funktion *DdeInitialize* angibt. *DdeCallback* ist ein Platzhalter für den anwendungs- oder bibliotheksdefinierten Funktionsnamen. Der eigentliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Anwendung exportiert werden.

Siehe auch

DdeEnableCallback, DdeInitialize

DdeClientTransaction 3.1

Syntax

```
#include <ddeml.h>
HDDEDATA DdeClientTransaction(lpvData, cbData, hConv,
   hszItem, uFmt, uType, uTimeout, lpuResult)

void FAR* lpvData;    /* Adresse der Daten, die an
                         die Server-Anwendung
                         übergeben werden            */
DWORD cbData;         /* Länge der Daten             */
HCONV hConv;          /* Handle der Konversation     */
HSZ hszItem;          /* Handle des String mit dem
                         Elementnamen                */
UINT uFmt;            /* Datenformat der
                         Zwischenablage              */
UINT uType;           /* Typ der Transaktion         */
DWORD uTimeout;       /* Timeout-Wert                */
DWORD FAR* lpuResult; /* zeigt auf das Ergebnis der
                         Transaktion                 */
```

Beschreibung

Die Funktion *DdeClientTransaction* startet eine Datentransaktion zwischen einem Client und einem Server. Eine DDE-Client-Anwendung kann diese Funktion aufrufen, nachdem eine Konversation mit dem Server eingerichtet wurde.

Parameter

lpvData: Zeigt auf den Anfang der Daten, die der Client an den Server übergeben muß. Eine Anwendung kann optional das Daten-Handle (HDDEDATA) angeben, das dem Server übergeben werden soll. In diesem Fall sollte der Parameter *cbData* auf -1 gesetzt werden. Dieser Parameter wird nur benötigt, wenn der Parameter *uType XTYP_EXECUTE* oder *XTYP_POKE* ist. Andernfalls sollte dieser Parameter *NULL* sein.

cbData: Gibt die Länge der Daten in Byte an, auf die der Parameter *lpvData* zeigt. Ein Wert von -1 bedeutet, daß *lpv-*

Data ein Daten-Handle ist, welches die gesendeten Daten bezeichnet.

hConv: Bezeichnet die Konversation, in der die Transaktion stattfinden soll.

hszItem: Bezeichnet das Datenelement, dessen Daten während der Transaktion ausgetauscht werden sollen. Dieses Handle muß vorher durch einen Aufruf der Funktion *DdeCreateStringHandle* erzeugt werden. Der Parameter wird ignoriert und sollte auf *NULL* gesetzt sein, wenn der Parameter *uType XTYP_EXECUTE* ist.

uFmt: Gibt das voreingestellte Format der Zwischenablage an, in dem das Datenelement geliefert oder angefordert wird.

uType: Gibt den Transaktionstyp an. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
XTYPF_ACKREQ	Weist den Server an, solange zu warten, bis der Client bestätigt, das vorherige Datenelement erhalten zu haben, bevor das nächste Datenelement gesendet wird. Dieses Flag hindert schnelle Server-Anwendungen daran, Daten schneller zu senden, als der Client die Daten bearbeiten kann.
XTYP_ADVSTART	Startet eine Advise-Schleife. Innerhalb einer Konversation können beliebig viele verschiedene Advise-Schleifen vorhanden sein. Eine Anwendung kann durch Kombination des Transaktionstyps *XTYP_ADVSTART* mit einem oder mehreren der folgenden Flags die Advise-Schleife verändern:
XTYP_ADVSTOP	Beendet eine Advise-Schleife.
XTYP_EXECUTE	Startet eine Execute-Transaktion.
XTYPF_NODATA	Weist den Server an, dem Client jede Datenveränderung anzuzeigen, ohne die Daten tatsächlich zu senden. Dieses Flag eröffnet dem Client die Wahl, die Benachrichtigung zu ignorieren oder die veränderten Daten vom Server anzufordern.
XTYP_POKE	Startet eine Poke-Transaktion.
XTYP_REQUEST	Startet eine Request-Transaktion.

uTimeout: Gibt die maximale Zeit in Millisekunden an, die die Client-Anwendung in einer synchronen Transaktion auf eine

Antwort der Server-Anwendung wartet. Dieser Parameter sollte für asynchrone Transaktionen auf den Wert *TIMEOUT_ASYNC* gesetzt werden.

lpuResult: Zeigt auf eine Variable, die das Ergebnis der Transaktion aufnimmt. Eine Anwendung, die das Ergebnis nicht überprüft, kann diesen Wert auf *NULL* setzen. Bei synchronen Transaktionen enthält das niederwertige *WORD* dieser Variablen ein *DDE*-Flag, das sich aus der Transaktion ergibt. Dies unterstützt jene Anwendungen, die vom Bit *DDE_APPSTATUS* abhängig sind. Anwendungen sollten diese Bits nicht mehr benutzen, da sie in zukünftigen Versionen der DDE-Bibliothek möglicherweise nicht mehr unterstützt werden. Bei asynchronen Transaktionen wird diese Variable mit einem eindeutigen Transaktionsbezeichner belegt.

Dieser Bezeichner kann zusammen mit der Funktion *DdeAbandonTransaction* und der Transaktion *XTYP_XACT_COMPLETE* benutzt werden.

Rückgabewert

Bei einer erfolgreichen synchronen Transaktion, in der der Client Daten vom Server erwartet, ist der Rückgabewert ein Handle, das die Daten bezeichnet. Der Rückgabewert ist *TRUE*, wenn asynchrone Transaktionen erfolgreich waren. Bei synchronen Transaktionen, in denen die Client-Anwendung keine Daten erwartet, ist der Rückgabewert ebenfalls *TRUE*. Bei allen fehlgeschlagenen Transaktionen ist der Rückgabewert *FALSE*.

Fehler

Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_ADVACKTIMEOUT
DMLERR_BUSY
DMLERR_DATAACKTIMEOUT
DMLERR_DLL_NOT_INITIALIZED
DMLERR_EXECACKTIMEOUT
DMLERR_INVALIDPARAMETER
DMLERR_MEMORY_ERROR
DMLERR_NO_CONV_ESTABLISHED
DMLERR_NO_ERROR
DMLERR_NOTPROCESSED
DMLERR_POKEACKTIMEOUT
DMLERR_POSTMSG_FAILED
DMLERR_REENTRANCY
DMLERR_SERVER_DIED
DMLERR_UNADVACKTIMEOUT

Hinweis

Wenn die Anwendung das von der Funktion *DdeClientTransaction* zurückgegebene Daten-Handle nicht mehr benötigt, sollte sie dieses Handle durch Aufruf der Funktion *DdeFreeDataHandle* freigeben.

Transaktionen können synchron oder asynchron sein. Während einer synchronen Transaktion kehrt die Funktion *DdeClientTransaction* solange nicht zurück, bis die Transaktion erfolgreich abgeschlossen wird oder fehlschlägt. Synchrone Transaktionen veranlassen die Client-Anwendung in eine modale Schleife einzutreten, während sie auf asynchrone Ereignisse wartet. Deshalb kann die Client-Anwendung zwar auf Benutzereingaben antworten, während sie auf die synchrone Transaktion wartet. Sie kann aber wegen der Aktivitäten der ersten Transaktion keine zweite synchrone Transaktion starten. Die Funktion *DdeClientTransaction* schlägt fehl, wenn eine Instanz derselben Anwendung bereits eine synchrone Transaktion bearbeitet.

Während einer asynchronen Transaktion kehrt die Funktion *DdeClientTransaction* nach dem Start der Transaktion zurück und übergibt einen Transaktionsbezeichner als Referenz. Wenn die Callback-Funktion der Server-Anwendung die Bearbeitung einer asynchronen Transaktion beendet, sendet das System eine *XTYP_XACT_COMPLETE*-Transaktion an den Client. Diese Transaktion versorgt den Client mit den Ergebnissen der asynchronen Transaktion, die er durch den Aufruf der Funktion *DdeClientTransaktion* initialisiert hat. Eine Client-Anwendung kann eine asynchrone Transaktion durch Aufruf der Funktion *DdeAbandonTransaction* abbrechen.

Beispiel

Das folgende Beispiel fordert eine Advise-Schleife mit einer DDE-Server-Anwendung an:

```
HCONV hconv;
HSZ hszNow;
HDDEDATA hData;
DWORD dwResult;

hData = DdeClientTransaction(
    (LPBYTE) NULL,      /* Keine Daten an den Server
                           übergeben */
    0,                  /* keine Daten */
    hszNow,             /* Elementname */
    CF_TEXT,            /* Format der Zwischenablage */
XTYP_ADVSTART,          /* Starten einer Advise-
                           Schleife */
1000,                   /* Timeout nach einer Sekunde */
&dwResult);             /* zeigt auf die Ergebnis-
                           Flags */
```

Siehe auch

DdeAbandonTransaction, DdeAccessData, DdeConnect, DdeConnectList, DdeCreateStringHandle

DdeCmpStringHandles 3.1

Syntax

```
#include <ddeml.h>
int DdeCmpStringHandles(hsz1, hsz2)

HSZ hsz1          /* Handle des ersten String       */
HSZ hsz2          /* Handle des zweiten String      */
```

Beschreibung Die Funktion *DdeCmpStringHandles* vergleicht den Wert zweier String-Handles. Der Wert eines String-Handle steht in keinem Zusammenhang mit der Groß- und Kleinschreibung des zugehörigen String.

Parameter *hsz1:* Bestimmt das erste String-Handle.

hsz2: Bestimmt das zweite String-Handle.

Rückgabewert Möglich sind folgende Rückgabewerte:

Wert	Bedeutung
-1	Der Wert von *hsz1* ist entweder 0 oder kleiner als der Wert von *hsz2*.
0	Die Werte von *hsz1* und *hsz2* sind gleich (beide können 0 sein).
1	Der Wert von *hsz2* ist entweder 0 oder kleiner als der Wert von *hsz1*.

Hinweis Anwendungen, die einen Vergleich zweier String-Handles benötigen, in dem Groß- und Kleinschreibung berücksichtigt werden, sollten die String-Handles direkt vergleichen. Für alle anderen Vergleiche sollten Anwendungen die Funktion *DdeCmpStringHandles* verwenden, um die voreingestellte Berücksichtigung der Schreibweise beim dynamischen Datenaustausch (DDE) zu erhalten.

Die Funktion *DdeCmpStringHandles* kann nicht dazu verwendet werden, String-Handles alphabetisch zu sortieren.

Beispiel Dieses Beispiel vergleicht die String-Handles zweier Servicenamen. Sind die Handles gleich, wird eine Konversation mit dem Server angefordert und eine *XTYP_ADVSTART*-Transaktion ausgegeben:

```
HSZ hszClock;     /* Servicename */
HSZ hszTime;      /* Themenname */
HSZ hsz1;         /* unbekannter Server */
HCONV hConv;      /* Handle der Konversation */
DWORD dwResult;   /* Ergebnis-Flags */
DWORD idInst;     /* Instanzenbezeichner */
```

```
/*
 * Das Handle eines unbekannten Servicenamens mit dem
 * String-Handle der Windows-Anwendung Uhr vergleichen
 */

if (!DdeCmpStringHandles(hsz1, hszClock)) {

    /*
     * Handelt es sich um die Uhr-Anwendung, wird eine
     * Konversation mit ihr gestartet und eine
     * Advise-Schleife angefordert.
     */

    hConv = DdeConnect(idInst, hszClock, hszTime, MULL);
    if (hConv != (HCONV) NULL)
        DdeClientTransaction(NULL, 0, hConv, hszNow,
            CF_TEXT, XTYP_ADVSTART, 1000, &dwResult);
}
```

Siehe auch *DdeAccessData, DdeCreateStringHandle, DdeFreeStringHandle*

DdeConnect 3.1

Syntax

```
#include <ddeml.h>
HCONV DdeConnect(idInst, hszService, hszTopic, pCC)

DWORD idInst;           /* Instanzenbezeichner        */
HSZ hszService;         /* Handle des String mit
                           dem Servicenamen           */
HSZ hszTopic;           /* Handle des String mit
                           dem Themennamen            */
CONVCONTEXT FAR* pCC;   /* Adresse der Struktur mit
                           Kontextdaten               */
```

Beschreibung Die Funktion *DdeConnect* richtet eine Konversation mit einer Server-Anwendung ein, die das angegebenene Paar Servicename und Themenname unterstützt. Wenn mehr als eine solche Server-Anwendung vorhanden ist, wählt das System nur eine davon aus.

Parameter *idInst:* Bestimmt den Bezeichner der Anwendungsinstanz, der durch einen früheren Aufruf der Funktion *DdeInitialize* ermittelt wurde.

hszService: Bezeichnet den String, der den Servicenamen der Server-Anwendung bestimmt, mit der eine Konversation eingerichtet werden soll. Dieses Handle muß durch einen vorhergehenden Aufruf der Funktion *DdeCreateStringHandle* erzeugt worden sein. Wenn dieser Parameter den Wert *NULL* hat, so wird mit jedem verfügbaren Server eine Konversation eingerichtet.

hszTopic: Bezeichnet den String, der den Namen des Themas enthält, zu dem die Konversation eingerichtet werden soll. Dieses Handle muß durch einen vorhergehenden Aufruf der Funktion *DdeCreateStringHandle* erzeugt worden sein. Wenn dieser Parameter den Wert *NULL* hat, wird eine Konversation über jedes Thema eingerichtet, das vom ausgewähltem Server unterstützt wird.

pCC: Zeigt auf die *CONVCONTEXT*-Struktur, die Information über den Kontext der Konversation enthält. Wenn dieser Parameter *NULL* ist, empfängt die Server-Anwendung während der *XTYP_CONNECT*- oder *XTYP_WILDCONNECT*-Transaktion die voreingestellte *CONVCONTEXT*-Struktur.

Die *CONVCONTEXT*-Struktur hat folgende Form:

```
#include <ddeml.h>

typedef struct tagCONVCONTEXT {  /* cc */
    UINT cb;
    UINT wFlags;
    UINT wCountryID;
    int iCodePage;
    DWORD dwLangID;
    DWORD dwSecurity;
} CONVCONTEXT;
```

Rückgabewert

Wenn die Funktion erfolgreich war, ist der Rückgabewert das Handle der eingerichteten Konversation. Andernfalls ist er *NULL*.

Fehler

Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_DLL_NOT_INITIALIZED
DMLERR_INVALIDPARAMETER
DMLERR_NO_CONV_ESTABLISHED
DMLERR_NO_ERROR

Hinweis

Die Client-Anwendung sollte keine Annahmen hinsichtlich der Auswahl eines Server machen. Wenn ein instanzenspezifischer Name im Parameter *hszService* angegeben wurde, wird die Konversation nur mit dieser Instanz eingerichtet. Instanzenspezifische Servicenamen werden während der *XTYP_REGISTER*- und *XTYP_UNREGISTER*-Transaktionen an die DDE-Callback-Funktion einer Anwendung übergeben.

Alle Elemente der voreingestellten *CONVCONTEXT*-Struktur sind mit Null initialisiert. Ausgenommen sind *cb* und *iCodePage*. *cb* bestimmt die Größe der Struktur. *iCodePage* enthält *CP_WINANSI*, die Standard-Code-Seite.

Beispiel

Das folgende Beispiel erzeugt String-Handles für einen Service- und einen Themennamen. Schlägt die anschließende Einrich-

tung einer Konversation mit einem Server fehl, der den Service- und den Themennamen unterstützt, wird ein Fehlerwert abgerufen, der den Grund des Fehlers bestimmt:

```
DWORD idInst = 0L;
HSZ hszClock;
HSZ hszTime;
HCONV hconv;
UINT uError;

hszClock = DdeCreateStringHandle(idInst, "Clock",
   CP_WINANSI);
hszTime = DdeCreateStringHandle(idInst, "Time",
   CP_WINANSI);

if ((hconv = DdeConnect(
   idInst,            /* Instanzenbezeichner */
   hszClock,          /* Servicename des Servers */
   hszTime,           /* Themenname */
   NULL)) == NULL) {  /* voreingestellte CONVCONTEXT-
                         Struktur benutzen */
   uError = DdeGetLastError(idInst);
}
```

Siehe auch *DdeConnectList, DdeCreateStringHandle, DdeDisconnect, DdeDisconnectList, DdeInitialize*

DdeConnectList 3.1

Syntax

```
#include <ddeml.h>
HCONVLIST DdeConnectList(idInst, hszService, hszTopic,
   hConvList, pCC)

DWORD idInst;          /* Instanzenbezeichner        */
HSZ hszService;        /* Handle des String mit
                          dem Servicenamen           */
HSZ hszTopic;          /* Handle des String mit
                          dem Themennamen            */
HCONVLIST hConvList;   /* Handle der
                          Konversationsliste         */
CONVCONTEXT FAR* pCC;  /* Adresse der Struktur
                          mit den Kontextdaten       */
```

Beschreibung Die Funktion *DdeConnectList* richtet eine Konversation mit allen Server-Anwendungen ein, die das angegebene Paar Service-/Themenname unterstützen. Eine Anwendung kann diese Funktion auch verwenden, um eine Liste der Konversations-Handles aufzuzählen, indem ein existierendes Konversations-Handle an die Funktion übergeben wird. Bei der Auflistung entfernt die DDE-Bibliothek (DDEML) die Handles beendeter Konversationen aus der Konversationsliste. Die resultierende

Konversationsliste enthält die Handles aller aktuell eingerichteten Konversationen, die den angegebenen Service- und Themennamen unterstützen.

Parameter

idInst: Bestimmt den Bezeichner der Anwendungsinstanz, der durch einen vorhergehenden Aufruf der Funktion *DdeInitialize* ermittelt wurde.

hszService: Bezeichnet den String, der den Servicenamen der Server-Anwendung bestimmt, mit der die Konversation eingerichtet werden soll. Wenn dieser Parameter den Wert *NULL* hat, wird eine Konversation mit allen verfügbaren Server-Anwendungen eingerichtet, die den angegebenen Themennamen unterstützen.

hszTopic: Bezeichnet den String, der den Namen des Themas enthält, zu dem die Konversation eingerichtet werden soll. Dieses Handle muß durch einen vorhergehenden Aufruf der Funktion *DdeCreateStringHandle* erzeugt worden sein. Wenn dieser Parameter den Wert *NULL* hat, wird eine Konversation zu allen Themen eingerichtet, die vom ausgewählten Server unterstützt werden.

hConvList: Bezeichnet die Konversationsliste, die aufgezählt werden soll. Dieser Parameter sollte auf *NULL* gesetzt werden, wenn eine neue Konversationsliste eingerichtet werden soll.

pCC: Zeigt auf die *CONVCONTEXT*-Struktur, die Information über den Konversationskontext enthält. Wenn dieser Parameter *NULL* ist, erhält die Server-Anwendung während der *XTYP_CONNECT*- oder *XTYP_WILDCONNECT*-Transaktionen die voreingestellte *CONVCONTEXT*-Struktur.

Die *CONVCONTEXT*-Struktur hat folgende Form:

```
#include <ddeml.h>

typedef struct tagCONVCONTEXT {  /* cc */
    UINT cb;
    UINT wFlags;
    UINT wCountryID;
    int iCodePage;
    DWORD dwLangID;
    DWORD dwSecurity;
} CONVCONTEXT;
```

Rückgabewert

Wenn die Funktion erfolgreich war, ist der Rückgabewert das Handle einer neuen Konversationsliste. Andernfalls ist er *NULL*. Das Handle der alten Konversationsliste ist nicht mehr gültig.

Fehler

Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_DLL_NOT_INITIALIZED
DMLERR_INVALID_PARAMETER
DMLERR_NO_CONV_ESTABLISHED
DMLERR_NO_ERROR
DMLERR_SYS_ERROR

Hinweis

Eine Anwendung muß das von dieser Funktion zurückgegebene Handle freigeben, ohne Rücksicht darauf, ob ein Konversations-Handle dieser Liste aktiv ist. Um das Handle freizugeben, kann die Anwendung die Funktion *DdeDisconnectList* aufrufen.

Alle Elemente der voreingestellten *CONVCONTEXT*-Struktur außer *cb*, das die Größe der Struktur angibt und *iCodePage*, das *CP_WINANSI* (die Voreinstellung für die Code-Seite) enthält, werden auf Null gesetzt.

Beispiel

Das folgende Beispiel richtet mit Hilfe der Funktion *DdeConnectList* eine Konversation mit allen Server-Anwendungen ein, die das Thema System unterstützen. Nachdem die Server gezählt sind, wird ein Puffer für die String-Handles der Servicenamen der Server-Anwendungen reserviert, in den dann die Handles kopiert werden:

```
HCONVLIST hconvList;  /* Konversations-Handle */
DWORD idInst;         /* Instanzenbezeichner */
HSZ hszSystem;        /* Thema System */
HCONV hconv = NULL;   /* Handle der Konversation */
CONVINFO ci;          /* enthält Daten der
                         Konversation */
UINT cConv = 0;       /* Zähler für die Handles
                         der Konversation */
HSZ *pHsz, *aHsz      /* Zeiger auf die
                         String-Handles */

/*
 * Konversation mit allen Server-Anwendungen
 * einrichten, die das Thema System unterstützen.
 */

hconvList = DdeConnectList(idInst, (HSZ) NULL,
   hszSystem, (HCONV) NULL, (LPVOID) NULL);

/* Anzahl der Handles in der Konversationsliste
   ermitteln. */

while ((hconv = DdeQueryNextServer(hconvList, hconv))
   != (HCONV) NULL) cConv++;

/* Einen Puffer für die String-Handles reservieren. */

hconv = (HCONV) NULL;
aHsz = (HSZ *) LocalAlloc(LMEM_FIXED,
   cConv * sizeof(HSZ));
```

```
/* Die String-Handles in den Puffer kopieren. */

pHsz = aHsz;
while ((hconv = DdeQueryNextServer(hconvList, hconv))
   != (HCONV) NULL) {
    DdeQueryConvInfo(hconv, QID_SYNC, (PCONVINFO) &ci);
    DdeKeepStringHandle(idInst, ci.hszSvcPartner);
    *pHsz++ = ci.hszSvcPartner;
}

.
. /* Die Handles benutzen. */
.

/* Speicher freigeben und Konversation beenden. */

LocalFree((HANDLE) aHsz);
DdeDisconnectList(hconvList);
```

Siehe auch *DdeConnect, DdeCreateStringHandle, DdeDisconnect, DdeDisconnectList, DdeInitialize, DdeQueryNextServer*

DdeCreateDataHandle 3.1

Syntax

```
#include <ddeml.h>
HDDEDATA DdeCreateDataHandle(idInst, lpvSrcBuf,
   cbInitData, offSrcBuf, hszItem, uFmt, afCmd)

DWORD idInst;         /* Instanzenbezeichner          */
void FAR* lpvSrcBuf;  /* Adresse des Quellpuffers     */
DWORD cbInitData;     /* Länge des globalen
                         Speicherobjekts              */
DWORD offSrcBuf;      /* Offset ab dem Anfang des
                         Quellpuffers                 */
HSZ hszItem;          /* Handle des String für den
                         Elementnamen                 */
UNIT uFmt;            /* Datenformat der
                         Zwischenablage               */
UNIT afCmd;           /* Erzeugungs-Flags             */
```

Beschreibung Die Funktion *DdeCreateDataHandle* erzeugt ein globales Speicherobjekt und füllt dieses mit den Daten, auf die der Parameter *lpvSrcBuf* zeigt. DDE-Anwendungen benutzen diese Funktion während solcher Transaktionen, die Daten an die Partner-Anwendung übergeben.

Parameter *idInst:* Bestimmt den Bezeichner der Anwendungsinstanz, der durch einen vorhergehenden Aufruf der Funktion *DdeInitialize* ermittelt wurde.

lpvSrcBuf: Zeigt auf einen Puffer mit den Daten, die in den globalen Speicherbereich kopiert werden sollen. Wenn dieser

Parameter *NULL* ist, werden keine Daten in das Speicherobjekt kopiert.

cbInitData: Gibt die Menge des benötigten Speichers in Byte an, der für das globale Speicherobjekt reserviert werden soll. Wenn dieser Parameter Null ist, wird der Parameter *lpvSrcBuf* ignoriert.

offScrBuf: Bestimmt einen Offset in Byte, ausgehend vom Anfang des durch *lpvSrcBuf* bezeichneten Puffers. Die Daten werden ab diesem Offset in das globale Speicherobjekt kopiert.

hszItem: Bezeichnet den String, der das Datenelement bestimmt, das dem globalen Speicherobjekt zugeordnet ist. Dieses Handle muß durch einen vorhergehenden Aufruf der Funktion *DdeCreateStringHandle* erzeugt worden sein. Wenn das Daten-Handle in einer *XTYP_EXECUTE*-Transaktion verwendet werden soll, muß dieser Parameter auf den Wert *NULL* gesetzt werden.

uFmt: Bezeichnet das voreingestellte Format der Zwischenablage.

afCmd: Bezeichnet die Erzeugungs-Flags. Dieser Parameter kann *HDATA_APPOWNED* sein. In diesem Fall besitzt die Server-Anwendung, die die Funktion *DdeCreateDataHandle* aufruft, das von dieser Funktion erzeugte Daten-Handle. Dies ermöglicht dem Server, das Daten-Handle mit mehreren Client-Anwendungen gemeinsam zu nutzen, ohne daß bei jeder Anforderung ein neues Handle erzeugt werden muß. Wenn dieses Flag gesetzt ist, muß der Server das gemeinsam benutzte Speicherobjekt, das diesem Handle zugeordnet ist, schließlich freigeben. Dies geschieht durch Aufruf der Funktion *DdeFreeDataHandle*. Wenn dieses Flag nicht gesetzt ist, wird das Daten-Handle für die erzeugende Anwendung ungültig, nachdem es von der DDE-Callback-Funktion des Server zurückgegeben oder als Parameter in einer anderen Funktion der DDEML benutzt wurde.

Rückgabewert Wenn die Funktion erfolgreich war, ist der Rückgabewert ein Daten-Handle. Andernfalls ist er *NULL*.

Fehler Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_DLL_NOT_INITIALIZED
DMLERR_INVALID_PARAMETER
DMLERR_MEMORY_ERROR
DMLERR_NO_ERROR

Hinweis Alle nicht belegten Positionen im globalen Speicherobjekt sind undefiniert.

Nachdem ein Daten-Handle als Parameter in einer anderen DDEML-Funktion verwendet oder von einer DDE-Callback-Funktion zurückgeliefert wurde, darf es nur für Lesezugriffe auf das bezeichnete globale Speicherobjekt benutzt werden.

Wenn die Anwendung dem globalen Speicherobjekt durch Aufruf der Funktion *DdeAddData* Daten hinzufügen will und das Objekt dann eine Größe von 64 KByte überschreiten würde, sollte die Anwendung eine Gesamtlänge (*cbInitData* + *offSrcData*) angeben. Diese Gesamtlänge sollte der erwarteten maximalen Größe des Objekts entsprechen. Dies vermeidet ein unnötiges Kopieren von Daten und Speicherreservierungen durch das System.

Beispiel

Das folgende Beispiel verarbeitet die Transaktion *XTYP_WILDCONNECT* durch Rückgabe eines Daten-Handle für ein Array mit *HSZPAIR*-Strukturen. Das Array enthält für jeden unterstützten Themennamen eine Struktur:

```
#define CTOPICS 2

UINT type;
UINT fmt;
HSZPAIR ahp[(CTOPICS + 1)];
HSZ ahszTopicList[CTOPICS];
HSZ hszServ, hszTopic;
WORD i, j;

if (type == XTYP_WILDCONNECT) {

/* Themenliste überprüfen und ein Array mit
   HSZPAIR-Strukturen erzeugen. */

   j = 0;
   for (i = 0; i < CTOPICS; i++) {
        if (hszTopic == (HSZ) NULL ||
                hszTopic == ahszTopicList[i]) {
            ahp[j].hszSvc = hszServ;
            ahp[j++].hszTopic = ahszTopicList[i];
        }
   }

  /*
   * Die Liste wird mit einer HSZPAIR-Struktur beendet,
   * deren Elemente NULL-String-Handles enthalten.
   */

   ahp[j].hszSvc = NULL;
   ahp[j++].hszTopic = NULL;

  /*
   * Zurückgegeben wird ein Handle für ein globales
   * Speicherobjekt, das die HSZPAIR-Strukturen enthält
   */
```

```
    return DdeCreateDataHandle(
      idInst,           /* Instanzenbezeichner */
      &ahp,             /* zeigt auf ein Array mit
                           HSZPAIR-Strukturen */
      sizeof(HSZ) * j,  /* Länge des Array */
      0,                /* am Anfang beginnen */
      NULL,             /* kein String für
                           Elementnamen */
      fmt,              /* Rückgabe im gleichen Format */
      0);               /* Handle wird dem System
                           zugeordnet */
}
```

Siehe auch *DdeAccessData, DdeFreeDataHandle, DdeGetData, DdeInitialize*

DdeCreateStringHandle 3.1

Syntax

```
#include <ddeml.h>
HSZ DdeCreateStringHandle(idInst, lpszString, codepage)

DWORD idInst;       /* Instanzenbezeichner           */
LPCSTR lpszString;  /* Adresse des nullterminierten
                       String                        */
int codepage;       /* Code-Seite                    */
```

Beschreibung *DdeCreateStringHandle* erzeugt ein Handle, das den String bezeichnet, auf den der Parameter *lpszString* zeigt. Eine DDE-Server- oder Client-Anwendung kann dieses String-Handle als Parameter an andere DDEML-Funktionen übergeben.

Parameter *idInst:* Bestimmt den Bezeichner der Anwendungsinstanz, der durch einen vorhergehenden Aufruf der Funkton *DdeInitialize* ermittelt wurde.

lpszString: Zeigt auf einen Puffer, der einen nullterminierten String enthält, für den das Handle erzeugt werden soll. Dieser String kann eine beliebige Größe haben.

codepage: Gibt die Code-Seite an, die zur Übergabe des String benutzt wird. Dieser Wert sollte entweder *CP_WINANSI* oder der Rückgabewert der Funktion *GetKBCodePage* sein. Der Wert Null entspricht *CP_WINANSI.*

Rückgabewert Wenn die Funktion erfolgreich war, ist der Wert ein String-Handle. Andernfalls ist er *NULL.*

Fehler Verwenden Sie die Funktion *DdeGetLastError,* um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_INVALID_PARAMETER
DMLERR_NO_ERROR
DMLERR_SYS_ERROR

Hinweis

Zwei identische Strings entsprechen immer demselben String-Handle. String-Handles sind innerhalb aller DDEML-Tasks eindeutig. Wenn also eine Anwendung ein Handle für einen String und eine andere Anwendung ein Handle für einen identischen String erzeugt, sind die String-Handles, die an beide Anwendungen zurückgegeben werden, ebenfalls identisch. Groß- und Kleinschreibung wird bei der Feststellung der Identität nicht berücksichtigt.

Wenn eine Anwendung ein String-Handle erzeugt oder von einer Callback-Funktion ein String-Handle empfangen und die Funktion *DdeKeepStringHandle* aufgerufen hat, um dieses Handle zu behalten, muß sie es freigeben, sobald es nicht mehr benötigt wird.

Ein instanzenspezifisches String-Handle kann nicht auf einen String und anschließend wieder auf dasselbe String-Handle abgebildet werden. Dies wird im folgenden Beispiel gezeigt. Die Funktion *DdeQueryString* erzeugt einen String zu einem String-Handle. Anschließend erzeugt *DdeCreateStringHandle* ein String-Handle für diesen String, der dann nicht mit dem ersten Handle identisch ist:

```
DWORD idInst;
DWORD cb;
HSZ hszInst, hszNew;
PSZ pszInst;

DdeQueryString(idInst, hszInst, pszInst, cb,
   CP_WINANSI);
hszNew = DdeCreateStringHandle(idInst, pszInst,
   CP_WINANSI);

/* hszNew != hszInst ! */
```

Beispiel

Das folgende Beispiel erzeugt String-Handles für Service- und Themennamen. Anschließend wird versucht, eine Konversation mit einem Server einzurichten, der Service- und Themennamen unterstützt. Schlägt die Einrichtung fehl, wird ein Fehlerwert abgerufen, der den Grund des Fehlers bezeichnet:

```
DWORD idInst = 0L;
HSZ hszClock;
HSZ hszTime;
HCONV hconv;
UINT uError;

hszClock = DdeCreateStringHandle(idInst, "Clock",
   CP_WINANSI);
hszTime = DdeCreateStringHandle(idInst, "Time",
   CP_WINANSI);

if ((hconv = DdeConnect(
   idInst,              /* Instanzenbezeichner */
```

```
    hszClock,            /* Servicename der
                            Server-Anwendung */
    hszTime,             /* Themenname */
    NULL)) == NULL) {    /* voreingestellte
                            CONVCONTEXT-Struktur
                            benutzen */
    uError = DdeGetLastError(idInst);
}
```

Siehe auch *DdeAccessData, DdeCmpStringHandles, DdeFreeStringHandles, DdeInitialize, DdeKeepStringHandle, DdeQueryString*

DdeDisconnect 3.1

Syntax

```
#include <ddeml.h>
BOOL DdeDisconnect(hConv)

HCONV hconv;    /* Handle der Konversation          */
```

Beschreibung Die Funktion *DdeDisconnect* beendet eine Konversation, die entweder von der Funktion *DdeConnect* oder von *DdeConnectList* gestartet wurde. Das Konversations-Handle wird ungültig gemacht.

Parameter *hConv:* Bezeichnet die aktive Konversation, die beendet werden soll.

Rüchgabewert Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich war. Andernfalls ist er Null.

Fehler Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_DLL_NOT_INITIALIZED
DMLERR_NO_CONV_ESTABLISHED
DMLERR_NO_ERROR

Hinweis Alle unvollständigen Transaktionen, die vor dem Aufruf von *DdeDisconnect* gestartet wurden, werden sofort abgebrochen. Der Transaktionstyp *XTYP_DISCONNECT* wird an die DDE-Callback-Funktion des Konversationspartners gesendet. In der Regel müssen nur Client-Anwendungen Konversationen beenden.

Siehe auch *DdeConnect, DdeConnectList, DdeDisconnectList*

DdeDisconnectList 3.1

Syntax

```
#include <ddeml.h>
BOOL DdeDisconnectList(hConvList)

HCONVLIST hConvList; /* Handle der Konversationsliste */
```

Beschreibung Die Funktion *DdeDisconnectList* gibt die angegebene Konversationsliste frei und beendet alle Konversationen, die dieser Liste zugeordnet sind.

Parameter *hConvList:* Bezeichnet die Konversationsliste. Dieses Handle muß durch einen vorhergehenden Aufruf der Funktion *DdeConnectList* erzeugt worden sein.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich war. Andernfalls ist er Null.

Fehler Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_DLL_NOT_INITIALIZED
DMLERR_INVALIDPARAMETER

Hinweis Um einzelne Konversationen in der Liste zu beenden, kann eine Anwendung die Funktion *DdeDisconnect* aufrufen.

Siehe auch *DdeConnect, DdeConnectList, DdeDisconnect*

DdeEnableCallback 3.1

Syntax

```
#include <ddeml.h>
BOOL DdeEnableCallback(idInst, hConv, uCmd)

DWORD idInst;  /* Instanzenbezeichner                */
HCONV hConv;   /* Handle der Konversation            */
UNIT uCmd;     /* Funktionscode für
                  Aktivieren/Deaktivieren            */
```

Beschreibung Die Funktion *DdeEnableCallback* aktiviert oder deaktiviert Transaktionen für eine bestimmte oder für alle Konversationen, die die Anwendung aktuell eingerichtet hat.

Nachdem die Transaktionen für eine Konversation deaktiviert wurden, ordnet das System die Transaktionen dieser Konversation in eine der Anwendung zugeordnete Transaktionswarteschlange ein. Die Anwendung sollte die Konversation möglichst bald wieder aktivieren. Damit wird verhindert, daß Transaktionen in einer Warteschlange verloren gehen.

Parameter *idInst:* Bestimmt den Bezeichner der Anwendungsinstanz, der durch einen vorhergehenden Aufruf der Funkton *DdeInitialize* ermittelt wurde.

hConv: Bezeichnet die Konversation, die aktiviert oder deaktiviert werden soll.

uCmd: Gibt den Funktionscode an. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
EC_ENABLEALL	Aktiviert alle Transaktionen für die angegebene Konversation.
EC_ENABLEONE	Aktiviert eine Transaktion für die angegebene Konversation.
EC_DISABLE	Deaktiviert alle sperrbaren Transaktionen für die angegebene Konversation. Eine Server-Anwendung kann folgende Transaktionen deaktivieren: *XTYP_ADVSTART*, *XTYP_ADVSTOP*, *XTYP_EXECUTE*, *XTYP_POKE*, *XTYP_REQUEST* Eine Client-Anwendung kann folgende Transaktionen deaktivieren: *XTYP_ADVDATA*, *XTYP_XACT_COMPLETE*

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich war. Andernfalls ist er Null.

Fehler Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_DLL_NOT_INITIALIZED
DMLERR_NO_ERROR
DMLERR_INVALIDPARAMETER

Hinweis Ein Anwendung kann Transaktionen für eine bestimmte Konversation deaktivieren, indem ihre DDE-Callback-Funktion *CBR_BLOCK* zurückgibt. Wenn die Konversation durch die Verwendung der Funktion *DdeEnableCallback* dann wieder aktiviert wird, erzeugt das System die gleiche Transaktion, die bearbeitet wurde, als die Konversation noch aktiviert war.

Siehe auch *DdeConnect*, *DdeConnectList*, *DdeDisconnect*, *DdeInitialize*

DdeFreeDataHandle 3.1

Syntay

```
#include <ddeml.h>
BOOL Dde FreeDataHandle(hData)

HDDEDATA hData;          /* Handle des globalen
                            Speicherobjekts          */
```

Beschreibung Die Funktion *DdeFreeDataHandle* gibt ein globales Speicherobjekt frei und entfernt das dem Objekt zugeordnete Daten-Handle.

Parameter *hData:* Bezeichnet das globale Speicherobjekt, das freigegeben werden soll. Dieses Handle muß durch einen vorhergehenden

Aufruf der Funktion *DdeCreateDataHandle* erzeugt oder von der Funktion *DdeClientTransaction* zurückgeliefert worden sein.

Rückgabewert

Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich war. Andernfalls ist er Null.

Fehler

Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_INVALIDPARAMETER
DMLERR_NO_ERROR

Hinweis

Eine Anwendung muß *DdeFreeDataHandle* in folgenden Fällen aufrufen:

- Um ein globales Speicherobjekt freizugeben, das die Anwendung durch Aufruf der Funktion *DdeCreateDataHandle* reserviert hat, wenn das Daten-Handle niemals an eine andere Funktion der DDE-Bibliothek (DDEML) übergeben wurde.
- Um ein globales Speicherobjekt freizugeben, das die Anwendung unter Angabe des Flags *HDATA_APPOWNED* in einem Aufruf der Funktion *DdeCreateDataHandle* reserviert hat.
- Um ein globales Speicherobjekt freizugeben, dessen Handle die Anwendung durch einen Aufruf der Funktion *DdeClientTransaction* erhalten hat.

Das System gibt ein nicht zugeordnetes Objekt automatisch frei, wenn sein Handle von einer DDE-Callback-Funktion zurückgegeben oder als Parameter in einer DDEML-Funktion verwendet wurde.

Beispiel

Das folgende Beispiel erzeugt ein globales Speicherobjekt mit Hilfe-Informationen und gibt das Objekt wieder frei, nachdem dessen Handle an die Client-Anwendung übergeben wurde:

```
DWORD idInst;
HSZ hszItem;
HDDEDATA hDataHelp;

char szDdeHelp[] = "Hilfe zum DDEML-Test-Server:\r\n"\
 "Die Namen 'Service' (Server) und 'Test'\r\n"\
 "(Thema) können sich ändern. Folgende Elemente\r\n"\
 "werden vom Thema 'Test' unterstützt:\r\n"\
 "\tCount:\tWird bei Datenänderungen erhöht.\r\n"\
 "\tRand:\tWird bei jeder Datenänderung\r\n"\
 "\t\tverändert.\r\n"\
 "\t\tIm Runaway-Modus ändern sich obige Elemente\r\n"\
 "\t\tbei jeder Anforderung.\r\n"\
 "\tHuge:\tZufällig erzeugte Testdaten mit mehr\r\n"\
 "\t\tals 64 KByte. Der Test-Client kann diese\r\n"\
 "\t\tDaten überprüfen. Die Daten werden bei\r\n"\
 "\t\tjeder Anforderung neu berechnet. Dies\r\n"\
```

```
 "\t\tüberprüft gleichzeitig huge-Daten, die\r\n"\
 "\t\tder Client sendet oder bearbeitet.\r\n\"\
 "\tHelp:\tDiese Hilfe-Informationen. Die\r\n"\
 "\t\tDaten haben den Status APPOWNED.\r\n"\

/* Globales Speicherobjekt für die Hilfe-
   Informationen erzeugen. */

if (!hDataHelp) {
    hDataHelp = DdeCreateDataHandle(idInst, szDdeHelp,
        strlen(szDdeHelp) + 1, 0, hszItem, CF_TEXT,
        HDATA_APPOWNED);
}

.
. /* Hilfe-Informationen an die Client-
.    Anwendung übergeben. */
.

/* Das globale Speicherobjekt freigeben. */

if (hDataHelp)
    DdeFreeDataHandle(hDataHelp);
```

Siehe auch *DdeAccessData, DdeCreateDataHandle*

DdeFreeStringHandle 3.1

Syntax

```
#include <ddeml.h>
BOOL DdeFreeStringHandle(idInst, hsz)

DWORD idInst;          /* Instanzenbezeichner       */
HSZ hsz;               /* Handle des String         */
```

Beschreibung Die Funktion *DdeFreeStringHandle* gibt ein String-Handle in der aufrufenden Anwendung frei.

Parameter *idInst:* Bestimmt den Bezeichner der Anwendungsinstanz, der durch einen vorhergehenden Aufruf der Funkton *DdeInitialize* ermittelt wurde.

hsz: Gibt das String-Handle an, das freigegeben werden soll. Dieses Handle muß durch einen vorhergehenden Aufruf der Funkton *DdeCreateStringHandle* erzeugt worden sein.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich war. Andernfalls ist er Null.

Hinweis Eine Anwendung kann die String-Handles freigeben, die sie mit der Funktion *DdeCreateStringHandle* erzeugt. Sie sollte aber diejenigen nicht freigeben, die das System an die DDE-Callback-Funktion übergeben hat oder die von der Funktion *DdeQueryConvInfo* in einer *CONVINFO*-Struktur zurückgegeben wurden.

Beispiel

Das folgende Beispiel gibt String-Handles während einer *XTYP_DISCONNECT*-Transaktion frei:

```
DWORD idInst = 0L;
HSZ hszClock;
HSZ hszTime;
HSZ hszNow;
UINT type;

if (type == XTYP_DISCONNECT) {

    DdeFreeStringHandle(idInst, hszClock);
    DdeFreeStringHandle(idInst, hszTime);
    DdeFreeStringHandle(idInst, hszNow);

    return (HDDEDATA) NULL;
}
```

Siehe auch

DdeCmpStringHandles, DdeCreateStringHandle, DdeInitialize, DdeKeepStringHandle, DdeQueryString

DdeGetData 3.1

Syntax

```
#include <ddeml.h>
DWORD DdeGetData(hData, pDest, cbMax, offSrc)

HDDEDATA hData;   /* Handle des globalen
                     Speicherobjekts                */
void FAR* pDest;  /* Adresse des Zielpuffers        */
DWORD cbMax;      /* Menge der zu kopierenden Daten */
DWORD offSrc;     /* Offset des Datenanfangs        */
```

Beschreibung

Die Funktion *DdeGetData* kopiert Daten aus dem angegebenen globalen Speicherobjekt in einen lokalen Puffer.

Parameter

hData: Bezeichnet das globale Speicherobjekt, das die Daten enthält, die kopiert werden sollen.

pDest: Zeigt auf den Puffer, der die Daten erhält. Wenn dieser Parameter *NULL* ist, liefert die Funktion *DdeGetData* die Menge der Daten in Byte zurück, die in den Puffer kopiert werden.

cbMax: Gibt die maximale Menge der Daten in Byte an, die in den Puffer kopiert werden sollen, auf den der Parameter *pDest* zeigt.

offSrc: Bezeichnet einen Offset innerhalb des globalen Speicherobjekts. Die Daten werden ab diesem Offset aus dem Objekt kopiert.

Rückgabewert

Wenn der Parameter *pDest* auf einen Puffer zeigt, ist der Rückgabewert die Größe des Speicherobjekts in Byte, das dem Daten-Handle zugeordnet ist oder die Größe des Parameters *cbMax*, je nach dem, welcher Wert kleiner ist.

Wenn der Parameter *pDest NULL* ist, ist der Rückgabewert die Größe des Speicherobjekts in Byte, das dem Daten-Handle zugeordnet ist.

Fehler

Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_DLL_NOT_INITIALIZED
DMLERR_INVALID_HDDEDATA
DMLERR_INVALIDPARAMETER
DMLERR_NO_ERROR

Beispiel

Das folgende Beispiel kopiert Daten aus einem globalen Speicherobjekt in einen lokalen Puffer. Anschließend wird die *TIME*-Struktur mit den Daten aus diesem Puffer gefüllt:

```
HDDEDATA hData;
char szBuf[32];
typedef struct {
    int hour;
    int minute;
    int second;
} TIME;

DdeGetData(hData, (LPBYTE) SZBuf, 32L, 0L);
sscanf(szBuf, "%d:%d:%d", &nTime.hour, &nTime.minute,
    &nTime.second);
```

Siehe auch

DdeAccessData, DdeCreateDataHandle, DdeFreeDataHandle

DdeGetLastError 3.1

Syntax

```
#include <ddeml.h>
UNIT DdeGetLastError(idInst)

DWORD idInst;    /* Instanzenbezeichner              */
```

Beschreibung

Die Funktion *DdeGetLastError* liefert den letzten Fehlerwert zurück, der von einer Funktion der DDE-Bibliothek (DDEML) gesetzt wurde und setzt den Fehlerwert auf *DMLERR_NO_ERROR* zurück.

Parameter

idInst: Bestimmt den Bezeichner der Anwendungsinstanz, der durch einen vorhergehenden Aufruf der Funkton *DdeInitialize* ermittelt wurde.

Rückgabewert

Der Rückgabewert ist der letzte Fehlerwert. Möglich sind folgende DDEML-Fehlerwerte:

Wert	Bedeutung
DMLERR_ADVACK-TIMEOUT	Das Timeout einer Anfrage für eine synchrone Advise-Transaktion war abgelaufen.

Wert	Bedeutung
DMLERR_BUSY	Das *DDE_FBUSY*-Bit wurde als Antwort auf eine Transaktion gesetzt.
DMLERR_DATA-ACKTIMEOUT	Das Timeout einer Anfrage für eine synchrone Datentransaktion war abgelaufen.
DMLERR_DLL_-NOT_INITIALIZED	Eine DDEML-Funktion wurde aufgerufen, ohne daß vorher die Funktion *DdeInitialize* aufgerufen wurde, oder ein ungültiger Instanzenbezeichner ist an eine DDEML-Funktion übergeben worden.
DMLERR_DLL_-USAGE	Eine Anwendung, die als *APPCLASS_MONITOR* initialisiert wurde, hat versucht, eine DDE-Transaktion auszuführen, oder eine Anwendung, die als *APPCMD_CLIENTONLY* initialisiert wurde, hat versucht Server-Transaktionen auszuführen.
DMLERR_EXEC-ACKTIMEOUT	Das Timeout einer Anfrage für eine synchrone Execute-Transaktion war abgelaufen.
DMLERR_INVALID-PARAMETER	Ein Parameter konnte von der DDEML nicht gültig erklärt werden. Einige mögliche Gründe sind folgende: ∩ Die Anwendung benutzte ein Daten-Handle, das mit einem anderen Elementnamen-Handle initialisiert wurde, als die Transaktion benötigt. ∩ Die Anwendung benutzte ein Daten-Handle, das mit einem anderen Format für die Zwischenablage initialisiert wurde, als für die Transaktion erforderlich ist. ∩ Die Anwendung benutzte ein Konversations-Handle für den Client innerhalb einer Funktion für den Server oder umgekehrt. ∩ Die Anwendung benutzte ein bereits freigegebenes Daten- oder String-Handle. ∩ Mehr als eine Instanz der Anwendung benutzten dasselbe Objekt.
DMLERR_LOW_-MEMORY	Eine DDEML-Anwendung hat den Speicher aufgebraucht, weil eine Server-Anwendung laufend mehr Daten lieferte, als die Client-Anwendung verarbeiten konnte.
DMLERR_-MEMORY_ERROR	Eine Speicheranforderung ist fehlgeschlagen.

Wert	Bedeutung
DMLERR_NO_CONV_ESTABLISHED	Der Versuch der Einrichtung einer Konversation durch eine Client-Anwendung ist fehlgeschlagen.
DMLERR_NOT_PROCESSED	Eine Transaktion ist fehlgeschlagen.
DMLERR_POKEACK-TIMEOUT	Das Timeout einer Anfrage für eine synchrone Poke-Transaktion war abgelaufen.
DMLERR_POSTMSG_FAILED	Ein interner Aufruf der Funktion *PostMessage* schlug fehl.
DMLERR_REENTRANCY	Eine Anwendungsinstanz, die bereits eine synchrone Transaktion ausführte, versuchte eine weitere synchrone Transaktion einzurichten oder die Funktion *DdeEnableCallback* wurde innerhalb einer DDEML-Callback-Funktion aufgerufen.
DMLERR_SERVER_DIED	Eine Transaktion der Server-Seite wurde in einer Konversation gestartet, die bereits vom Client beendet worden war oder der Server wurde vor Abschluß der Transaktion beendet.
DMLERR_SYS_ERROR	Ein interner Fehler ist innerhalb der DDEML aufgetreten.
DMLERR_UNADVACKTIMEOUT	Das Timeout einer Anfrage zur Beendigung einer Advise-Transaktion war abgelaufen.
DMLERR_UNFOUND_QUEUE_ID	Ein ungültiger Transaktionsbezeichner wurde an eine DDEML-Funktion übergeben. Wenn die Anwendung von einem *XTYP_XACT_COMPLETE*-Callback zurückgekehrt ist, ist der Transaktionsbezeichner nicht mehr gültig.

Beispiel

Das folgende Beispiel ruft die Funktion *DdeGetLastError* auf, wenn die Funktion *DdeCreateDataHandle* fehlschlägt:

```
DWORD idInst;
HDDEDATA hddeMyData;
HSZPAIR ahszp[2];
HSZ hszClock, hszTime;

/* String-Handles erzeugen.  */

hszClock = DdeCreateStringHandle(idInst,
   (LPSTR) "Clock", CP_WINANSI);
```

```
hszTime = DdeCreateStringHandle(idInst,
   (LPSTR) "Time", CP_WINANSI);

/* Handles in eine HSZPAIR-Struktur kopieren. */

ahszp[0].hszSvc = hszClock;
ahszp[0].hszTopic = hszTime;
ahszp[1].hszSvc = (HSZ) NULL;
ahszp[1].hszTopic = (HSZ) NULL;

/* Ein globales Speicherobjekt erzeugen. */

hddeMyData = DdeCreateDataHandle(idInst, ahszp,
        sizeof(ahszp), 0, NULL, CF_TEXT, 0);
    if (hddeMyData == NULL)
        /*
         * Fehlerwert wird an eine anwendungsdefinierte
         * Funktion zur Fehlerbearbeitung übergeben
         */

        HandleError(DdeGetLastError(idInst));
```

Siehe auch *DdeInitialize*

DdeInitialize 3.1

Syntax

```
#include <ddeml.h>
UNIT DdeInitialize(lpidInst, pfnCallback, afCmd, uRes)

DWORD FAR* lpidInst;      /* Adresse des
                             Instanzenbezeichners     */
PFNCALLBACK pfnCallback;  /* Adresse der Callback-
                             Funktion                 */
DWORD afCmd;              /* Array der Befehls- und
                             Filter-Flags             */
DWORD uRes;               /* reserviert               */
```

Beschreibung Die Funktion *DdeInitialize* registriert eine Anwendung für die DDE-Bibliothek (DDEML). Eine Anwendung muß diese Funktion aufrufen, bevor sie irgendeine andere DDEML-Funktion aufruft.

Parameter *lpidInst:* Zeigt auf den Bezeichner der Anwendungsinstanz. Bei der Initialisierung sollte dieser Parameter auf 0L zeigen. Wenn die Funktion erfolgreich war, zeigt dieser Parameter auf den Instanzenbezeichner der Anwendung. Dieser Wert sollte als Parameter *idInst* an alle DDEML-Funktionen übergeben werden, die ihn benötigen. Benutzt eine Anwendung mehrere Instanzen der dynamischen Linkbibliothek DDEML, sollte sie eine andere Callback-Funktion für jede Instanz bereitstellen.

Wenn *lpidInst* auf einen Wert ungleich Null zeigt, führt dies zu einer Neuinitialisierung der DDEML. In diesen Fall muß

lpidInst auf einen gültigen Bezeichner einer Anwendungsinstanz zeigen.

pfnCallback: Zeigt auf die anwendungsdefinierte DDE-Callback-Funktion. Diese Funktion bearbeitet DDE-Transaktionen, die vom System gesendet werden. Weitere Informationen finden Sie in der Beschreibung der Funktion *DdeCallback*.

afCmd: Bezeichnet ein Array aus *APPCMD_*- und *CBF_*- Flags. Die *APPCMD_*-Flags übergeben zusätzliche Anweisungen an die Funktion *DdeInitialize*. Die *CBF_*-Flags setzen Filter, die bestimmte Transaktionstypen für die Callback-Funktion unterdrücken. Der Gebrauch dieser Flags erhöht die Geschwindigkeit einer DDE-Anwendung, indem sie unnötige Aufrufe der Callback-Funktion verhindern.

Dieser Parameter kann eine Kombination folgender Flags sein:

Flag	Bedeutung
APPCLASS_-MONITOR	Ermöglicht der Anwendung, die DDE-Aktivität im System zu überwachen. Dieses Flag ist zum Gebrauch in DDE-Überwachungsanwendungen gedacht. Indem die Anwendung eines oder mehrere Überwachungs-Flags mit dem Flag *APPCLASS_MONITOR* kombiniert, gibt sie die Typen der DDE-Aktivitäten an, die überwacht werden sollen. Eine eingehende Darstellung finden Sie im »Hinweis«.
APPCLASS_-STANDARD	Registriert die Anwendung als (nichtüberwachende) DDEML-Standardanwendung.
APPCMD_-CLIENTONLY	Verhindert, daß die Anwendung eine Server-Anwendung in einer DDE-Konversation wird. Die Anwendung kann nur eine Client-Anwendung sein. Dieses Flag schränkt den Ressourcenverbrauch durch die DDEML ein. Die Funktionalität des Flag *CBF_FAIL_-ALLSVRXACTIONS* bleibt erhalten.
APPCMD_-FILTERINITS	Verhindert, daß die DDEML Transaktionen vom Typ *XTYP_CONNECT* oder *XTYP_WILD-CONNECT* an die Anwendung sendet, bis die Anwendung ihre String-Handles erzeugt und ihren Servicenamen registriert oder die Filterung durch einen nachfolgenden Aufruf von *DdeNameService* oder *DdeInitialize* ausgeschaltet hat.

Flag	Bedeutung
	Dieses Flag ist immer gesetzt, wenn eine Anwendung die Funktion *DdeInitialize* zum ersten Mal aufruft, unabhängig davon, ob die Anwendung selbst dieses Flag gesetzt hat. In folgenden Aufrufen von *DdeInitialize* werden die Servicenamenfilter der Anwendung abgeschaltet, wenn das Flag nicht gesetzt wird.
CBF_FAIL_ALL-SVRXACTIONS	Verhindert, daß die Callback-Funktion Server-Transaktionen empfängt. Das System übergibt jedem Client, der eine Transaktion an diese Anwendung gesendet hat, *DDE_FNOT-PROCESSED*. Dieses Flag entspricht der Kombination aller *CBF_FAIL_*-Flags.
CBF_FAIL_ADVISES	Verhindert, daß die Callback-Funktion die Transaktion *XTYP_ADVSTART* und *XTYP_-ADVSTOP* empfängt. Das System übergibt jedem Client *DDE_FNOTPROCESSED*, der eine *XTYP_ADVSTART*- oder *XTYP_ADVSTOP*-Transaktion einem Server gesendet hat.
CBF_FAIL_-CONNECTIONS	Verhindert, daß die Callback-Funktion Transaktionen *XTYP_CONNECT* oder *XTYP_WILD-CONNECT* empfängt.
CBF_FAIL_-EXECUTES	Verhindert, daß die Callback-Funktion *XTYP_EXECUTE*-Transaktionen empfängt. Das System übergibt jedem Client *DDE_-FNOTPROCESSED*, der einem Server eine *XTYP_EXECUTE*-Transaktion gesendet hat.
CBF_FAIL_POKES	Verhindert, daß die Callback-Funktion *XTYP_POKE*-Transaktionen empfängt. Das System übergibt jedem Client *DDE_FNOT-PROCESSED*, der einem Server eine *XTYP_-POKE*-Transaktion gesendet hat.
CBF_FAIL_-REQUESTS	Verhindert, daß die Callback-Funktion *XTYP_REQUEST*-Transaktionen empfängt. Das System übergibt jedem Client *DDE_-FNOTPROCESSED*, der einem Server eine *XTYP_REQUEST*-Transaktion gesendet hat.
CBF_FAIL_SELF-CONNECTIONS	Verhindert, daß die Callback-Funktion *XTYP_CONNECT*-Transaktionen von einer Instanz der eigenen Anwendung empfängt. Dadurch wird verhindert, daß eine Anwendung eine DDE-Konversation mit ihrer eigenen Instanz errichtet.

Flag	Bedeutung
	Dieses Flag sollte eine Anwendung dann benutzen, wenn sie eine Kommunikation mit ihren Instanzen, nicht aber mit der aufrufenden Instanz benötigt.
CBF_SKIP_ALL-NOTIFICATIONS	Verhindert, daß die Callback-Funktion Benachrichtigungen empfängt. Dieses Flag entspricht einer Kombination aller *CBF_SKIP_*-Flags.
CBF_SKIP_CON-NECT_CONFIRMS	Verhindert, daß die Callback-Funktion *XTYP_CONNECT_CONFIRM*-Benachrichtigungen empfängt.
CBF_SKIP_-DISCONNECTS	Verhindert, daß die Callback-Funktion *XTYP_DISCONNECT*-Benachrichtigungen empfängt.
CBF_SKIP_-REGISTATIONS	Verhindert, daß die Callback-Funktion *XTYP_REGISTER*-Benachrichtigungen empfängt.
CBF_SKIP_-UNREGISTRATIONS	Verhindert, daß die Callback-Funktion *XTYP_UNREGISTER*-Benachrichtigungen empfängt.

uRes: Reserviert; muß auf 0L gesetzt sein.

Rückgabewert Folgende Rückgabewerte sind möglich:

DMLERR_DLL_USAGE
DMLERR_INVALIDPARAMETER
DMLERR_NO_ERROR
DMLERR_SYS_ERROR

Hinweis Eine Anwendung, die mehrere Instanzen der DDEML verwendet, darf die DDEML-Objekte nicht zwischen den Instanzen austauschen.

Eine DDE-Überwachungsanwendung sollte nicht versuchen, innerhalb derselben Anwendungsinstanz einen dynamischen Datenaustausch auszuführen (Konversationen einrichten, Transaktionen übergeben u.s.w.).

Eine synchrone Transaktion schlägt fehl und liefert den Fehlerwert *DMLERR_REENTRANCY*, wenn eine Instanz derselben Task bereits eine synchrone Transaktion ausführt.

Eine DDE-Überwachungsanwendung kann ein oder mehrere der folgenden Überwachungs-Flags mit dem Flag *APPCLASS_MONITOR* kombinieren, um die Typen der zu überwachenden DDE-Aktivität festzulegen:

Flag	Beschreibung
MF_CALLBACKS	Benachrichtigt die Callback-Funktion immer dann, wenn eine Transaktion an eine DDE-Callback-Funktion im System gesendet wird.
MF_CONV	Benachrichtigt die Callback-Funktion immer dann, wenn eine Konversation eingerichtet oder beendet wird.
MF_ERRORS	Benachrichtigt die Callback-Funktion immer dann, wenn ein DDE-Fehler aufgetreten ist.
MF_HSZ_INFO	Benachrichtigt die Callback-Funktion immer dann, wenn eine DDE-Anwendung den Verwendungszähler eines String-Handles erzeugt, freigibt oder erhöht oder wenn ein String-Handle durch Aufruf der Funktion *DdeUninitialize* freigegeben wird.
MF_LINKS	Benachrichtigt die Callback-Funktion immer dann, wenn eine Advise-Schleife gestartet oder beendet wird.
MF_POSTMSGS	Benachrichtigt die Callback-Funktion immer dann, wenn das System oder eine Anwendung eine DDE-Botschaft übergibt.
MF_SENDMSGS	Benachrichtigt die Callback-Funktion immer dann, wenn das System oder eine Anwendung eine DDE-Botschaft sendet.

Beispiel

Das folgende Beispiel ermittelt eine Adresse für die Prozedurinstanz einer DDE-Callback-Funktion und initialisiert anschließend die Anwendung für die DDEML.

```
DWORD idInst = 0L;
FARPROC lpDdeProc;
lpDdeProc = MakeProcInstance((FARPROC) DDECallback,
    hInst);
if (DdeInitialize((LPDWORD) &idInst,
        (PFNCALLBACK) lpDdeProc, APPCMD_CLIENTONLY, 0L))
    return FALSE;
```

Siehe auch

DdeClientTransaction, DdeConnect, DdeCreateDataHandle, DdeEnableCallback, DdeNameService, DdePostAdvise, DdeUninitialize

DdeKeepStringHandle 3.1

Syntax

```
#include<ddeml.h>
BOOL DdeKeepStringHandle(idInst, hsz)
```

Beschreibung Die Funktion *DdeKeepStringHandle* erhöht den Verwendungszähler des angegebenen Handle um 1. Mit Hilfe dieser Funktion kann eine Anwendung ein String-Handle sichern, das an eine DDE-Callback-Funktion übergeben wurde.

Normalerweise werden String-Handles, die an eine Callback-Funktion übergeben werden, bei der Rückkehr der Funktion gelöscht.

Parameter *idInst:* Bestimmt den Bezeichner der Anwendungsinstanz, der durch einen vorhergehenden Aufruf der Funktion *DdeInitialize* ermittelt wurde.

hsz: Bezeichnet das zu sichernde String-Handle.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Beispiel Das folgende Beispiel ist ein Teil einer DDE-Callback-Funktion, die die Verwendungszähler zweier String-Handles erhöht und so lokale Kopien dieser Handles sichert.

```
HSZ hsz1;
HSZ hsz2;
static HSZ hszServerBase;
static HSZ hszServerInst;
DWORD idInst;

case XTYP_REGISTER;

    /* String-Handles für spätere Verwendung sichern. */

    DdeKeepStringHandle(idInst, hsz1);
    DdeKeepStringHandle(idInst, hsz2);
    hszServerBase = hsz1;
    hszServerInst = hsz2;
    .
    . /* Bearbeitung der Transaktion beenden */
    .
```

Siehe auch *DdeCreateStringHandle, DdeFreeStringHandle, DdeInitialize, DdeQueryString*

DdeNameService 3.1

Syntax

```
#include <ddeml.h>
HDDEDATA DdeNameService(idInst, hsz1, hszRes, afCmd)

DWORD idInst;  /* Instanzenbezeichner                */
HSZ hsz1;      /* Handle des String mit dem
                  Servicenamen                       */
HSZ hszRes;    /* reserviert                         */
UNIT afCmd;    /* Servicenamen-Flags                 */
```

Beschreibung

Die Funktion *DdeNameService* registriert Servicenamen oder hebt deren Registrierung auf, die von einer DDE-Server-Anwendung unterstützt werden. Diese Funktion bewirkt, daß das System *XTYP_REGISTER*- oder *XTYP_UNREGISTER*-Transaktionen an andere laufende DDE-Client-Anwendungen sendet.

Eine Server-Anwendung sollte diese Funktion aufrufen, um jeden Servicenamen zu registrieren, den sie unterstützt und um die Registrierung von Servicenamen aufzuheben, die sie nicht mehr unterstützt. Eine Server-Anwendung sollte diese Funktion auch aufrufen, bevor sie beendet wird, um die Registrierung ihrer Servicenamen aufzuheben.

Parameter

idInst: Gibt den Bezeichner der Anwendungsinstanz an, der durch einen vorhergehenden Aufruf der Funkton *DdeInitialize* ermittelt wurde.

hsz1: Bezeichnet den String, der den Servicenamen bestimmt, den der Server registrieren oder dessen Registrierung aufgehoben werden soll. Eine Anwendung, die die Registrierung aller Servicenamen aufheben will, sollte diesen Parameter auf *NULL* setzen.

hszRes: Reserviert; sollte auf *NULL* gesetzt sein.

afCmd: Bestimmt die Servicenamen-Flags. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
DNS_REGISTER	Registriert den angegebenen Servicenamen.
DNS_UNREGISTER	Hebt die Registrierung des angegebenen Servicenamens auf.
DNS_FILTERON	Aktiviert das Filtern von Servicenamen. Dieser Filter verhindert, daß ein Server *XTYP_WILDCONNECT*-Transaktionen für nicht registrierte Servicenamen empfängt. Dies ist die Voreinstellung für diesen Filter. Hat ein Server keinen Servicenamen registriert, so kann die Anwendung keine *XTYP_WILDCONNECT*-Transaktionen empfangen.
DNS_FILTEROFF	Deaktiviert das Filtern von Servicenamen. Wenn dieses Flag gesetzt ist, empfängt der Server eine *XTYP_WILDCONNECT*-Transaktion, sobald eine andere DDE-Anwendung die Funktion *DdeConnect* aufruft. Der Servicename wird nicht berücksichtigt.

Rückgabewert

Wenn die Funktion erfolgreich war, ist der Rückgabewert ungleich Null. Andernfalls ist er Null.

Fehler

Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_DLL_NOT_INITIALIZED
DMLERR_DLL_USAGE
DMLERR_INVALIDPARAMETER
DMLERR_NO_ERROR

Hinweis

Der Servicename, der durch den Parameter *hsz1* bezeichnet wird, sollte ein Basisname sein, also keine instanzenspezifischen Informationen enthalten. Das System erzeugt einen instanzenspezifischen Namen und sendet ihn während der *XTYP_REGISTER*- und *XTYP_UNREGISTER*-Transaktionen zusammen mit dem Basisnamen. Die empfangenden Anwendungen können sich dann mit der jeweiligen Anwendungsinstanz verbinden.

Beispiel

Das folgende Beispiel initialisiert eine Anwendung für die DDEML, erzeugt häufig benutzte String-Handles und registriert den Servicenamen der Anwendung:

```
HSZ hszClock;
HSZ hszTime;
HSZ hszNow;
HINSTANCE hinst;
DWORD idInst = 0L;
FARPROC lpDdeProc;

/* Anwendung für die DDEML initialisieren. */

lpDdeProc = MakeProcInstance((FARPROC) DdeCallback,
   hinst);
if (!DdeInitialize((LPDWORD) &idInst,
   (PFNCALLBACK) lpDdeProc, APPCMD_FILTERINITS |
   CBF_FAIL_EXECUTES, 0L)) {

   /* Häufig verwendete String-Handles erzeugen. */

   hszTime = DdeCreateStringHandle(idInst, "Time",
      CP_WINANSI);
   hszNow = DdeCreateStringHandle(idInst, "Now",
      CP_WINANSI);
   hszClock = DdeCreateStringHandle(idInst, "Clock",
      CP_WINANSI);

   /* Servicenamen registrieren. */

   DdeNameService(idInst, hszClock, (HSZ) NULL,
      DNS_REGISTER);

}
```

Siehe auch

DdeConnect, DdeConnectList, DdeInitialize

DdePostAdvise 3.1

Syntax

```
#include <ddeml.h>
BOOL DdePostAdvise(idInst, hszTopic, hszItem)

DWORD idInst;        /* Instanzenbezeichner          */
HSZ hszTopic;        /* Handle des String mit dem
                        Themennamen                  */
HSZ hszItem;         /* Handle des String mit dem
                        Elementnamen                 */
```

Beschreibung

Die Funktion *DdePostAdvise* veranlaßt das System, eine *XTYP_ADVREQ*-Transaktion an die DDE-Callback-Funktion der aufrufenden Server-Anwendung zu senden. Dies erfolgt für jede Client-Anwendung, die aktuell für das angegebene Paar aus Themen- und Elementname eine Advise-Schleife aktiviert hat.

Eine Server-Anwendung sollte diese Funktion immer dann aufrufen, wenn sich die Daten verändern, die dem Paar aus Themen- und Elementname zugeordnet sind.

Parameter

idInst: Gibt den Bezeichner der Anwendungsinstanz an, der durch einen vorhergehenden Aufruf der Funkton *DdeInitialize* ermittelt wurde.

hszTopic: Bezeichnet den String, der den Themennamen enthält. Um für alle Themen Benachrichtigungen senden zu lassen, für die eine Advise-Schleife aktiviert ist, kann eine Anwendung diesen Parameter auf *NULL* setzen.

hszItem: Bezeichnet den String, der den Elementnamen enthält. Um für alle Elemente Benachrichtigungen senden zu lassen, für die eine Advise-Schleife aktiviert ist, kann eine Anwendung diesen Parameter auf *NULL* setzen.

Rückgabewert

Wenn die Funktion erfolgreich war, ist der Rückgabewert ungleich Null. Andernfalls ist er Null.

Fehler

Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_DLL_NOT_INITIALIZED
DMLERR_DLL_USAGE
DMLERR_NO_ERROR

Hinweis

Eine Anwendung, die nichtauflistbare Elemente oder Themen hat, sollte die Parameter *hszTopic* und *hszItem* auf *NULL* setzen. Dann erzeugt das System Transaktionen für alle aktiven Advise-Schleifen. Die DDE-Callback-Funktion liefert für jede Advise-Schleife, die nicht aktualisiert werden muß, den Wert *NULL* zurück.

Wenn ein Server die Funktion *DdePostAdvise* mit einer Kombination aus Themen-/Element-/Formatnamen aufruft, die der

aktuell in einem *XTYP_ADVREQ*-Callback bearbeiteten entspricht, kann dies einen Stack-Überlauf zur Folge haben.

Beispiel

Das folgende Beispiel ruft die Funktion *DdePostAdvise* auf, sobald sich die Zeit ändert:

```
typedef struct { /* tm */
    int hour;
    int minutes;
    int second;
} TIME;

TIME tmTime;
DWORD idInst;
HSZ hszTime;
HSZ hszNow;
TIME tmCurTime;

     .
     .  /* tmCurTime wird mit der aktuellen Zeit
     .     initialisiert. */
     .

/*
 * änderungen der Uhrzeit - Sekunden, Minuten oder
 * Stunden - werden überprüft.
 */

if ((tmCurTime.second != tmTime.second) ||
        (tmCurTime.minute != tmTime.minute) ||
        (tmCurTime.hour   != tmTime.hour)) {

    /* Die aktuelle Zeit wird an die Client-
       Anwendung übergeben. */

    DdePostAdvise(idInst, hszTime, hszNow);
```

Siehe auch

DdeInitialize

DdeQueryConvInfo 3.1

Syntax

```
#include <ddeml.h>
UNIT DdeQueryConvInfo(hConv, idTransaction, lpConvInfo)

HCONV hConv;              /* Handle der Konversation  */
DWORD idTransaction;      /* Transaktionsbezeichner   */
CONVINFO FAR* lpConvInfo; /* Adresse der Struktur, die
                             die Konversationsdaten
                             aufnimmt                 */
```

Beschreibung

Die Funktion *DdeQueryConvInfo* liefert Informationen über eine DDE-Transaktion und über die Konversation, in der die Transaktion stattgefunden hat.

Parameter

hConv: Bezeichnet die Konversation.

idTransaction: Gibt die Transaktion an. Dieser Parameter sollte für asynchrone Transaktionen ein Transaktionsbezeichner sein, der von der Funktion *DdeClientTransaction* zurückgeliefert wurde. Für synchrone Transaktionen sollte dieser Parameter den Wert *QID_SYNC* haben.

lpConvInfo: Zeigt auf eine *CONVINFO*-Struktur, die die Informationen über die Transaktion und die Konversation aufnimmt. Das Element *cb* der *CONVINFO*-Struktur muß die Länge des Puffers angeben, der für die Struktur reserviert wird.

Die *CONVINFO*-Struktur hat folgende Form:

```
typedef tagCONVINFO {  /*  ci  */
    DWORD cb;
    DWORD hUser;
    HCONV hConvPartner;
    HSZ hszSvcPartner;
    HSZ hszServiceReq;
    HSZ hszTopic;
    HSZ hszItem;
    UNIT wFmt;
    UNIT wType;
    UNIT wStatus;
    UNIT wConvst;
    UNIT wLastError;
    HCONVLIST hConvList;
    CONVCONTEXT ConvCtxt;
} CONVINFO;
```

Rückgabewert

Wenn die Funktion erfolgreich war, ist der Rückgabewert die Anzahl der Byte, die in die *CONVINFO*-Struktur kopiert wurden. Andernfalls ist er Null.

Fehler

Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_DLL_NOT_INITIALIZED
DMLERR_NO_CONV_ESTABLISHED
DMLERR_NO_ERROR
DMLERR_UNFOUND_QUEUE_ID

Beispiel

Das folgende Beispiel füllt eine *CONVINFO*-Struktur mit Informationen über eine synchrone Konversation und ermittelt dann die Namen der Partneranwendung und des Themas:

```
DWORD idInst;
HCONV hConv;
CONVINFO ci;
WORD wError;
char szSvcPartner[32];
```

```
char szTopic[32];
DWORD szServ, cchTopic;

if (!DdeQueryConyInfo(hConv, QID_SYNC, &ci))
    wError = DdeGetLastError(idInst);

else {
    cchServ = DdeQueryString(idInst, ci.hszSvcPartner,
        (LPSTR) &szSvcPartner, sizeof(szSvcPartner),
        CP_WINANSI);
    cchTopic = DdeQueryString(isInst, ci.hszTopic,
        (LPSTR) &szTopic, sizeof(szTopic), CP_WINANSI);
}
```

Siehe auch *DdeConnect, DdeConnectList, DdeQueryNextServer*

DdeQueryNextServer 3.1

Syntax

```
#include <ddeml.h>
HCONV DdeQueryNextServer(hConvList, hConvPrev)

HCONVLIST hConvList; /* Handle der Konversationsliste */
HCONV hConvPrev;     /* vorhergehendes Konversations-
                        Handle                        */
```

Beschreibung Die Funktion *DdeQueryNextServer* ermittelt das nächste Konversations-Handle in der angegebenen Konversationsliste.

Parameter *hConvList:* Bezeichnet die Konversationsliste. Dieses Handle muß durch einen vorhergehenden Aufruf der Funktion *DdeConnectList* erzeugt worden sein.

hConvPrev: Bezeichnet das Konversations-Handle, das zuvor von dieser Funktion zurückgeliefert wurde. Wenn dieser Parameter *NULL* ist, liefert die Funktion das erste Konversations-Handle in der Liste zurück.

Rückgabewert Wenn die Liste weitere Konversations-Handles enthält, ist der Rückgabewert das nächste Konversations-Handle in der Liste. Andernfalls ist er Null.

Beispiel Im folgenden Beispiel wird die Anzahl Konversations- Handles in einer Konversationsliste ermittelt. Dann werden die String-Handles der Servicenamen in einen lokalen Puffer kopiert:

```
HCONVLIST hconvList; /* Konversationsliste */
DWORD idInst;        /* Instanzenbezeichner */
HSZ hszSystem;       /* Thema System */
HCONV hconv = NULL;  /* Handle der Konversation */
CONVINFO ci;         /* enthält Konversationsdaten */
UINT cConv = 0;      /* Zähler für die Konversations-
                        Handles */
HSZ *pHsz, *aHsz;    /* zeigt auf die String-Handles */
```

```
/* Es wird eine Verbindung mit allen Server-Anwendungen
   hergestellt, die das Thema unterstützen. */

hconvList = DdeConnectList(idInst, (HSZ) NULL,
    hszSystem, (HCONV) NULL, (LPVOID) NULL);

/* Die Anzahl der Handles in der Konversationsliste
   wird ermittelt. */

while ((hconv = DdeQueryNextServer(hconvList, hconv))
    != (HCONV) NULL) cConv++;

/* Ein Puffer für die String-Handles wird reserviert. */

hconv = (HCONV) NULL;
aHsz = (HSZ *) LocalAlloc(LMEM_FIXED,
            cConv * sizeof(HSZ));

/* String-Handles werden in den Puffer kopiert. */

pHsz = aHsz;
while ((hconv = DdeQueryNextServer(hconvList, hconv))
      != (HCONV) NULL) {
    DdeQueryConvInfo(hconv, QID_SYNC, (PCONVINFO) &ci);
    DdeKeepStringHandle(idInst, ci.hszSvcPartner);
    *pHsz++ = ci.hszSvcPartner;
}

. /*  Die Handles werden für die Konversation mit den
.  *  Server-Anwendungen benutzt.
.  */

/* Der Speicher wird freigegeben, die Konversationen
   werden beendet.  */

LocalFree(HANDLE);
DdeDisconnectList(hconvList);
```

Siehe auch *DdeConnect, DdeDisconnectList*

DdeQueryString 3.1

Syntax

```
#include <ddeml.h>
DWORD DdeQueryString(idInst, hsz, lpsz, cchMax,
   codepage)

DWORD idInst;         /* Instanzenbezeichner         */
HSZ hsz;              /* Handle des String           */
LPSTR lpsz;           /* Adresse des Ziel-Puffers    */
DWORD cchMax;         /* Länge des Puffers           */
int codepage;         /* Code-Seite                  */
```

Beschreibung Die Funktion *DdeQueryString* kopiert einen Text, der einem String-Handle zugeordnet ist, in einen Puffer. Der String ist

immer nullterminiert. Wenn der String länger als (*cchMax* - 1) ist, werden nur die ersten (*cchMax* - 1) Zeichen des String kopiert.

Wenn der Parameter *lpsz NULL* ist, ermittelt diese Funktion die Länge des dem String-Handle zugeordneten String. Die Länge enthält das nullterminierende Zeichen nicht.

Parameter

idInst: Bestimmt den Bezeichner der Anwendungsinstanz, der durch einen vorhergehenden Aufruf der Funktion *DdeInitialize* ermittelt wurde.

hsz: Bezeichnet den String, der kopiert werden soll. Dieses Handle muß durch einen vorhergehenden Aufruf der Funktion *DdeCreateStringHandle* erzeugt worden sein.

lpsz: Zeigt auf einen Puffer, der den String aufnimmt. Um die Länge des String zu ermitteln, sollte dieser Parameter auf *NULL* gesetzt werden.

cchMax: Gibt die Länge des Puffers in Byte an, auf den der Parameter *lpsz* zeigt. Wenn der String länger als (*cchMax* - 1) ist, wird er abgeschnitten. Wenn der Parameter *lpsz NULL* ist, wird dieser Parameter ignoriert.

codepage: Bestimmt die Code-Seite, die zur Übergabe des String benutzt wird. Dieser Wert sollte entweder *CP_WINANSI* oder der Wert sein, den die Funktion *GetKBCodePage* zurückliefert.

Rückgabewert

Wenn der Parameter *lpsz* ein gültiger Zeiger ist, ist der Rückgabewert die Länge des zurückgelieferten Texts in Byte (das terminierende *NULL*-Zeichen nicht eingeschlossen). Der Rückgabewert ist die Länge des Textes (das terminierende *NULL*-Zeichen nicht eingeschlossen), der dem Parameter *hsz* zugeordnet ist, wenn der Parameter *lpsz* ein *NULL*-Zeiger ist. Wenn ein Fehler aufgetreten ist, ist der Rückgabewert *NULL*.

Beispiel

Das folgende Beispiel ermittelt die registrierten Service- und Themennamen einer Server-Anwendung mit Hilfe der Funktion *DdeQueryString*:

```
UINT type;

HSZ hsz1;
HSZ hsz2;
char szBaseName[16];
char szInstName[16];

if (type == XTYP_REGISTER) {

/* Der eigentliche Servicename wird in einen
   Puffer kopiert. */

   DdeQueryString(idInst, hsz1, (LPSTR) &szBaseName,
        sizeof(szBaseName), CP_WINANSI);
```

```
/* Der instanzenspezifische Servicename wird in einen
   Puffer kopiert. */

   DdeQueryString(idInst, hsz2, (LPSTR) &szInstName,
        sizeof(szInstName), CP_WINANSI);
    return (HDDEDATA) TRUE;
}
```

Siehe auch *DdeCmpStringHandles, DdeCreateStringHandle, DdeFreeStringHandle, DdeInitialize*

DdeReconnect 3.1

Syntax

```
#include <ddeml.h>

HCONV DdeReconnect(hConv)
HCONV hConv;       /* Handle der Konversation, die
                      wiederhergestellt werden soll   */
```

Beschreibung Die Funktion *DdeReconnect* erlaubt einer DDEML-Client-Anwendung die Wiederherstellung einer Konversation mit einem Server, der die Konversation mit dem Client beendet hat. Wenn die Konversation wiederhergestellt ist, versucht die DDEML alle vorher existierenden Advise-Schleifen wiederherzustellen.

Parameter *hConv:* Bezeichnet die Konversation, die wiederhergestellt werden soll. Ein Client muß das Konversations-Handle durch einen vorhergehenden Aufruf der Funktion *DdeConnect* erhalten haben.

Rückgabewert Wenn die Funktion erfolgreich war, ist der Rückgabewert das Handle der wiederhergestellten Konversation. Der Rückgabewert ist *NULL*, wenn die Funktion fehlgeschlagen ist.

Fehler Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_DLL_NOT_INITIALIZED
DMLERR_INVALIDPARAMETER
DMLERR_NO_CONV_ESTABLISHED
DMLERR_NO_ERROR

Beispiel Das folgende Beispiel demonstriert eine Situation, in der eine Anwendung die Funktion *DdeReconnect* aufrufen sollte:

```
HDDEDATA EXPENTRY DdeCallback(wType, wFmt, hConv, hsz1,
    hsz2, hData, dwData1, dwData2)
WORD wType;      /* Transaktionstyp */
WORD wFmt;       /* Format der Zwischenablage */
HCONV hConv;     /* Handle der Konversation */
HSZ hsz1;        /* Handle eines String */
HSZ hsz2;        /* Handle eines String */
```

```
HDDEDATA hData;  /* Handle eines globalen
                    Speicherobjekts */
DWORD dwData1;   /* transaktionsspezifische Daten */
DWORD dwData2;   /* transaktionsspezifische Daten */
{
    BOOL fAutoReconnect;

    switch (wType) {
        case XTYP_DISCONNECT:
            if (fAutoReconnect) {
                DdeReconnect(hConv);
                 /*  Versuch der Wiederherstellung */
            }
            return 0;

        .
        .       /* Hier werden andere
        .          Transaktionen bearbeitet. */

    }
}
```

Siehe auch *DdeConnect, DdeDisconnectList*

DdeSetUserHandle 3.1

Syntax

```
#include <ddeml.h>
BOOL DdeSetUserHandle(hConv, id, hUser)

HCONV hConv;         /* Handle der Konversation       */
DWORD id;            /* Transaktionsbezeichner        */
DWORD hUser;         /* anwendungsdefinierter Wert    */
```

Beschreibung Die Funktion *DdeSetUserHandle* ordnet einem Konversations-Handle und einem Transaktionsbezeichner einen anwendungsdefinierten 32-Bit-Wert zu. Dadurch kann die Bearbeitung asynchroner Transaktionen vereinfacht werden. Um diesen Wert abzurufen, kann eine Anwendung die Funktion *DdeQueryConvInfo* verwenden.

Parameter *hConv:* Bezeichnet die Konversation.

id: Gibt den Transaktionsbezeichner einer asynchronen Transaktion an. Die Anwendung sollte diesem Parameter den Wert *QUI_SYNC* zuweisen, wenn dem Parameter *hUser* keine asynchrone Transaktion zugeordnet werden soll.

hUser: Bezeichnet den Wert, der dem Konversations-Handle zugeordnet werden soll.

Rückgabewert Wenn die Funktion erfolgreich war, ist der Rückgabewert ungleich Null. Andernfalls ist er Null.

Fehler Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_DLL_NOT_INITIALIZED
DMLERR_INVALIDPARAMETER
DMLERR_NO_ERROR
DMLERR_UNFOUND_QUEUE_ID

Siehe auch *DdeQueryConvInfo*

DdeUnaccessData 3.1

Syntax

```
#include <ddeml.h>

BOOL DdeUnaccessData(hData)
HDDEDATA hData;         /* Handle des globalen
                           Speicherobjekts          */
```

Beschreibung Die Funktion *DdeUnaccessData* gibt ein globales Speicherobjekt frei. Hat eine Anwendung den Zugriff auf ein Objekt abgeschlossen, muß sie diese Funktion aufrufen.

Parameter *hData:* Bezeichnet das globale Speicherobjekt.

Rückgabewert Wenn die Funktion erfolgreich war, ist der Rückgabewert ungleich Null. Andernfalls ist er Null.

Fehler Verwenden Sie die Funktion *DdeGetLastError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

DMLERR_DLL_NOT_INITIALIZED
DMLERR_INVALIDPARAMETER
DMLERR_NO_ERROR

Beispiel Das folgende Beispiel ruft einen Zeiger auf ein globales Speicherobjekt ab und kopiert mit dessen Hilfe Daten aus dem Objekt in einen lokalen Puffer. Der Aufruf der Funktion *DdeUnaccessData* gibt das Objekt frei:

```
HDDEDATA hData;
LPBYTE lpszAdviseData;
DWORD cbDataLen;
DWORD i;
char szData[128];

lpszAdviseData = DdeAccessData(hData, &cbDataLen);
for (i = 0; i < cbDataLen; i++)
    szData[i] = *lpszAdviseData++;
DdeUnaccessData(hData);
```

Siehe auch *DdeAccessData, DdeAddData, DdeCreateDataHandle, DdeFreeDataHandle*

DdeUninitialize 3.1

Syntax

```
#include <ddeml.h>
BOOL DdeUninitialize(idInst)

DWORD                /* Instanzenbezeichner          */
```

Beschreibung Die Funktion *DdeUninitialize* gibt alle DDEML-Ressourcen frei, die der aufrufenden Anwendung zugeordnet sind.

Parameter *idInst:* Bestimmt den Bezeichner der Anwendungsinstanz, der durch einen vorhergehenden Aufruf der Funktion *DdeInitialize* ermittelt wurde.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *DdeUninitialize* beendet jede aktuell geöffnete Konversation der Anwendung.

Beendet ein Konversationspartner seine Konversation nicht, tritt das System möglicherweise in eine modale Schleife ein, während es auf die Beendigung wartet. Dieser Schleife ist ein Timeout-Wert zugeordnet. Nach Ablauf dieser Zeit wird ein Meldungsfenster angezeigt. In diesem kann der Benutzer auswählen, ob er eine weitere Timeout-Periode abwarten (*Wiederholen*), unbegrenzt warten (*Ignorieren*) oder die modale Schleife verlassen (*Abbrechen*) will.

Anwendungen sollten diese Funktion erst aufrufen, wenn ihre Fenster nicht mehr sichtbar und ihre Botschaftsschleifen beendet sind.

Siehe auch *DdeDisconnect, DdeDisconnectList, DdeInitialize*

DebugOutput 3.1

Syntax

```
void FAR _cdecl DebugOutput(flags, lpsz, ...)

UINT flags           /* Botschaftstyp               */
LPCSTR lpsz          /* Zeiger auf Format-String    */
```

Beschreibung Die Funktion *DebugOutput* übergibt eine Botschaft an das Debug-Terminal.

Anwendungen können dem Botschafts-String formatierende Anweisungen hinzufügen, außerdem können sie Filter und Optionen verwenden, die den Botschaftstyp bestimmen.

Parameter *flags:* Bestimmt den Typ der Botschaft, die an das Debug-Terminal gesendet werden soll.

Dieser Parameter kann einen der in der folgenden Tabelle beschriebenen Werte annehmen:

Wert	Bedeutung
DBF_TRACE	Es sind keine Fehler aufgetreten. Außerdem stellt die Botschaft zusätzliche Informationen bereit, die beim Testen hilfreich sein können. Beispiel: »KERNEL: Lade SAMPLE.DLL«.
DBF_WARNING	Die Botschaft wird in einer Situation übergeben, in der – in Abhängigkeit von den jeweiligen Umständen – ein Fehler aufgetreten sein könnte. Beispiel: »KERNEL: LoadString fehlgeschlagen«.
DBF_ERROR	Ein Fehler ist wegen eines fehlgeschlagenen Aufrufs einer Windows-Funktion aufgetreten. Die Anwendung läuft weiter. Beispiel: »KERNEL: Ungültiger lokaler Heap«.
DBF_FATAL	Es ist ein Fehler aufgetreten, der zum Abbruch der Anwendung führt. Beispiel: »USER: Veraltete Funktion SetDeskWallPaper aufgerufen«.

lpsz: Zeigt auf einen Format-String, der denen der Windows-Funktion *wsprintf* entspricht. Dieser String muß weniger als 160 Zeichen enthalten. Zusätzliche Formatierungen können mit Hilfe weiterer Parameter durchgeführt werden, die dem Parameter *lpsz* folgen.

...: Optionale zusätzliche Argumente. Anzahl und Typ der Argumente hängt von den entsprechenden Formatsequenzen ab, die im Parameter *lpsz* angegeben sind.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Hinweis Die von der Funktion *DebugOutput* gesendeten Botschaften werden von den Debug-Optionen des Systems und den Filter-Flags für die Ablaufverfolgung beeinflußt, die mit den Funktionen *GetWinDebugInfo* und *SetWinDebugInfo* abgerufen und gesetzt werden. Die Optionen und Flags werden in einer *WINDEBUGINFO*-Struktur gespeichert.

Im Unterschied zu den meisten anderen Windows-Funktionen verwendet *DebugOutput* die C-Aufrufkonventionen (**_cdecl**) und nicht die Pascal-Aufrufkonventionen. Deshalb muß der Aufrufer die Argumente vom Stack nehmen. Außerdem werden die Argumente von rechts nach links auf den Stack gelegt. In C-Modulen übernimmt der Compiler diese Aufgabe.

Anwendungen müssen diese Funktion als importierte Funktion deklarieren. Folgende Zeile muß im IMPORTS-Abschnitt der Moduldefinitionsdatei der Anwendung stehen:

```
IMPORTS
    kernel._DebugOutput
```

Siehe auch *GetWinDebugInfo, OutputDebugString, SetWinDebugInfo, wsprintf*

DebugProc 3.1

Syntax

```
LRESULT CALLBACK DebugProc(code, wParam, lParam)

int code;           /* Filterfunktion                */
WPARAM wParam;      /* Typ des aufzurufenden Filters */
LPARAM lParam;      /* Adresse der Struktur, die
                       Debug-Information enthält     */
```

Beschreibung Die Funktion *DebugProc* ist eine bibliotheksdefinierte Callback-Funktion, die vor allen anderen Filtern, die durch die Funktion *SetWindowsHookEx* installiert wurden, vom System aufgerufen wird. Das System übergibt die Informationen über den aufzurufenden Filter an die Callback-Funktion *DebugProc*. Diese kann nun die Information prüfen und den Aufruf des Filters zulassen oder zurückweisen.

Parameter *code:* Gibt den Filter-Code an. Zur Zeit ist *HC_ACTION* der einzige gültige, positive Wert. Wenn dieser Parameter kleiner als Null ist, muß die Callback-Funktion ohne weitere Bearbeitung die Funktion *CallNextHookEx* aufrufen.

wParam: Bezeichnet das Task-Handle der Task, die den Filter installiert hat, der aufgerufen werden soll.

lParam: Enthält einen Zeiger vom Typ **long** auf eine *DEBUGHOOKINFO*-Struktur. Die *DEBUGHOOKINFO*-Struktur hat folgende Form:

```
typedef struct tagDEBUGHOOKINFO {
    HMODULE hModuleHook;
    LPARAM  reserved;
    LPARAM  lParam;
    WPARAM  wParam;
    int     code;
} DEBUGHOOKINFO;
```

Rückgabewert Die Callback-Funktion sollte *TRUE* zurückliefern, um das System am Aufruf anderer Filter zu hindern. Andernfalls muß die Callback-Funktion die Filterinformation an die Funktion *CallNextHookEx* übergeben.

Hinweis Eine Anwendung muß diese Callback-Funktion durch einen Aufruf der Funktion *SetWindowsHookEx* installieren, indem sie den Filtertyp *WH_DEBUG* und die Adresse der Prozedurinstanz der Callback-Funktion angibt.

CallWndProc ist ein Platzhalter für den bibliotheksdefinierten Funktionsnamen. Der eigentliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Bibliothek exportiert werden.

Siehe auch *CallNextHookEx, SetWindowsHookEx*

DefDriverProc 3.1

Syntax

```
LRESULT DdeDriverProc(dwDriverIdentifier, hdrvr, uMsg,
   lParam1, lParam2)

DWORD dwDriverIdentifier;  /* Bezeichner des installier-
                              baren Treibers          */
HDRVR hdrvr;               /* Handle des installier-
                              baren Treibers          */
UNIT uMsg;                 /* Nummer der Botschaft    */
LPARAM lParam1;            /* erster Botschafts-
                              parameter               */
LPARAM lParam2;            /* zweiter Botschafts-
                              parameter               */
```

Beschreibung Die Funktion *DefDriverProc* ermöglicht eine Standardbearbeitung aller Botschaften, die nicht von einem installierbaren Treiber bearbeitet werden.

Parameter *dwDriverIdentifier:* Bezeichnet einen installierbaren Treiber. Dieser Parameter muß durch einen vorhergehenden Aufruf der Funktion *OpenDriver* ermittelt worden sein.

hdrvr: Bezeichnet den installierbaren Treiber.

uMsg: Bestimmt die Botschaft, die bearbeitet werden soll.

lParam1: Bestimmt 32 Bits zusätzlicher botschaftsabhängiger Informationen.

lParam2: Bestimmt 32 Bits zusätzlicher botschaftsabhängiger Informationen.

Rückgabewert Wenn die Funktion erfolgreich war, ist der Rückgabewert ungleich Null. Andernfalls ist er Null.

Hinweis Die Funktion *DefDriverProc* bearbeitet Botschaften, die von der Funktion *DriverProc* nicht bearbeitet werden.

Siehe auch *OpenDriver, SendDriverMessage*

DirectedYield 3.1

Syntax

```
void DirectedYield(htask)

HTASK htask;      /* Handle der zu aktivierenden Task */
```

Beschreibung Die Funktion DirectedYield deaktiviert die aktuelle und aktiviert die angegebene Task.

Parameter *htask*: Bezeichnet die Task, die ausgeführt werden soll.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Hinweis Wenn ein unter Windows arbeitender Debugger die Steuerung an andere Anwendungen übergibt (d.h. den Hard-Modus verläßt), sollte er *DirectedYield* aufrufen. Dabei muß er das Handle der Task angeben. Das stellt sicher, daß diese Anwendung ausgeführt wird und daß Botschaften, die während des Testens empfangen werden, von den entsprechenden Fenstern bearbeitet werden.

Der Scheduler von Windows führt eine Task nur dann aus, wenn ein Ereignis (z.B. eine Zeichnen-Botschaft) oder eine übergebene Botschaft in der Botschaftswarteschlange für diese Task vorliegt.

Wenn eine Anwendung *DirectedYield* für eine Task verwendet, für die keine Ereignisse vorliegen, wird die Task nicht ausgeführt. Windows durchsucht stattdessen die Task-Warteschlange. Eine Anwendung kann aber jederzeit die Ausführung einer bestimmten Task veranlassen, indem sie die Funktion *PostAppMessage* aufruft und dabei den Botschaftsbezeichner *WM_NULL* angibt. Wenn die Anwendung anschließend *DirectedYield* aufruft, führt der Scheduler die Task ungeachtet ihres Ereignisstatus aus.

DirectedYield nimmt die durch *htask* bezeichnete Task an der Stelle wieder auf, an der sie verlassen wurde. Debugger sollten *TaskSwitch* anstelle von *DirectedYield* verwenden, da *TaskSwitch* eine Task an jeder gewünschten Adresse starten kann.

DirectedYield kehrt zurück, wenn die aktuelle Task wieder aktiviert ist. Dies tritt ein, wenn die durch *htask* bezeichnete Task auf Botschaften wartet oder die Funktion *DirectedYield* oder *Yield* verwendet.

DirectedYield befindet sich in den Dateien KRNL286.EXE und KRNL386.EXE und ist in den Windows-Versionen 3.0 und 3.1 verfügbar.

Siehe auch *PostAppMessage, TaskSwitch, TaskGetCSIP, TaskSetCSIP, Yield*

DlgDirSelectComboBoxEx 3.0

Syntax

```
BOOL DlgSelectComboBoxEx(hwndDlg, lpszPath, cbPath,
   idComboBox)

HWND hwndDlg;          /* Handle des Dialogfensters
                          mit der Liste              */
LPSTR lpszPath;        /* Adresse des Puffers für
                          den Verzeichnis-String     */
```

```
int cbPath;           /* Anzahl der Byte im
                         Verzeichnis-String        */
int idComboBox;       /* Bezeichner des
                         Kombinationsfensters      */
```

Beschreibung Die Funktion *DlgDirSelectComboBoxEx* ermittelt die aktuelle Auswahl in der Liste eines Kombinationsfensters. Die Liste sollte durch die Funktion *DlgDirListComboBox* gefüllt worden sein. Die Auswahl sollte ein Laufwerksbuchstabe, eine Datei oder ein Verzeichnisname sein.

Parameter *hwndDlg:* Gibt das Dialogfenster an, in dem sich das Kombinationsfenster befindet.

lpszPath: Zeigt auf einen Puffer, der das ausgewählte Verzeichnis oder den Dateinamen erhält.

cbPath: Bestimmt die Länge des Verzeichnisses oder Dateinamens, auf den der Parameter *lpszPath* zeigt, in Byte. Dieser Wert darf nicht größer als 128 Byte sein.

idComboBox: Der Integer-Bezeichner des Kombinationsfensters im Dialogfenster.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die aktuelle Auswahl im Kombinationsfenster ein Verzeichnisname ist. Andernfalls ist er Null.

Hinweis Die Funktion *DlgDirSelectComboBoxEx* läßt nur die Rückgabe eines einzelnen Dateinames aus einem Kombinationsfenster zu.

Wenn die aktuelle Auswahl ein Verzeichnisname oder ein Laufwerksbuchstabe ist, entfernt *DlgDirSelectComboBoxEx* die einschließenden eckigen Klammern (und Bindestriche bei Laufwerksbuchstaben), so daß der Name in einen neuen Verzeichnisnamen oder Dateinamen eingefügt werden kann. Wenn keine Auswahl getroffen wurde, verändert sich der Inhalt des durch den Parameter *lpszPath* bezeichneten Puffers nicht.

DlgDirSelectComboBoxEx sendet *CB_GETCURSEL*- und *CB_GETLBTEXT*-Botschaften an das Kombinationsfenster.

Siehe auch *DlgDirList, DlgDirListComboBox, DlgDirSelect, DlgDirSelectEx, DlgSirSelectComboBox*

DlgDirSelectEx 2.x

Syntax

```
BOOL DlgDirSelectEx(hwndDlg, lpszPath, cbPath,
   idListBox)

HWND hwndDlg;         /* Handle des Dialogfensters
                         mit der Liste             */
```

```
LPSTR lpszPath;       /* Adresse des Puffers für
                         den Verzeichnis-String      */
int cbPath;           /* Anzahl der Byte im
                         Verzeichnis-String          */
int idComboBox;       /* Bezeichner der Liste        */
```

Beschreibung Die Funktion *DlgDirSelectEx* ermittelt die aktuelle Auswahl in einer Liste. Die angegebene Liste sollte durch die Funktion *DlgDirList* gefüllt worden sein. Die Auswahl sollte ein Laufwerksbuchstabe, ein Datei- oder ein Verzeichnisname sein.

Parameter *hwndDlg:* Bezeichnet das Dialogfenster, in dem sich das Kombinationsfenster befindet.

lpszPath: Zeigt auf einen Puffer, der den ausgewählten Verzeichnis- oder Dateinamen aufnimmt.

cbPath: Bezeichnet die Länge des Verzeichnis- oder Dateinamens, auf den der Parameter *lpszPath* zeigt, in Byte. Dieser Wert darf nicht größer als 128 Byte sein.

idComboBox: Der Integer-Bezeichner der Liste im Dialogfenster.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die aktuelle Auswahl in der Liste ein Verzeichnisname ist. Andernfalls ist er Null.

Hinweis Wenn die aktuelle Auswahl ein Verzeichnisname oder ein Laufwerksbuchstabe ist, entfernt *DlgDirSelectEx* die einschließenden eckigen Klammern (und Bindestriche bei Laufwerksbuchstaben), so daß der Name in einen neuen Verzeichnis- oder Dateinamen eingefügt werden kann. Wenn keine Auswahl getroffen wurde, verändert sich der Inhalt des durch den Parameter *lpszPath* bezeichneten Puffers nicht.

Die Funktion *DlgDirSelectEx* läßt nur die Rückgabe eines einzelnen Dateinames aus einer Liste zu.

Die Liste darf keine Liste für Mehrfachauswahl sein. Wenn dies der Fall ist, liefert diese Funktion *NULL* zurück und *lpszPath* bleibt unverändert.

DlgDirSelectEx sendet *CB_GETCURSEL*- und *CB_GETLBTEXT*-Botschaften an die Liste.

Siehe auch *DlgDirList, DlgDirListComboBox, DlgDirSelect, DlgSirSelectComboBox*

DragAcceptFiles 3.1

Syntax

```
#include <shellapi.h>
void DragAcceptFiles(hwnd, fAccept)

HWND hwnd;            /* Fenster-Handle          */
```

```
BOOL fAccept;           /* Flag für die Entgegennahme
                           abgelegter Dateien       */
```

Beschreibung Die Funktion *DragAcceptFiles* registriert ein Fenster für die Entgegennahme abgelegter Dateien.

Parameter *hwnd:* Bezeichnet das Fenster, das für die Entgegennahme abgelegter Dateien registriert werden soll.

fAccept: Bestimmt, ob das durch den Parameter *hwnd* bezeichnete Fenster abgelegte Dateien entgegennimmt. Anwendungen sollten diesen Wert auf *TRUE* setzen, wenn abgelegte Dateien entgegengenommen werden sollen. Wird *FALSE* angegeben, werden abgelegte Dateien nicht akzeptiert.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Hinweis Wenn eine Anwendung die Funktion *DragAcceptFiles* mit dem Wert *TRUE* für den Parameter *fAccept* aufruft, sendet der Datei-Manager die Botschaft *WM_DROPFILES* an dieses Fenster, sobald der Benutzer dort eine Datei ablegt.

DragFinish 3.1

Syntax

```
#include <shellapi.h>
void DragFinish(hDrop)

HDROP hDrop;   /* Handle einer internen Datenstruktur */
```

Beschreibung Die Funktion *DragFinish* gibt Speicher frei, den Windows für die Übergabe von Dateinamen an die Anwendung reserviert hat.

Parameter *hDrop:* Bezeichnet die interne Datenstruktur, die abgelegte Dateien beschreibt. Dieses Handle wird im Parameter *wParam* der Botschaft *WM_DROPFILES* an die Anwendung übergeben.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

DragQueryFile 3.1

Syntax

```
#include <shellapi.h>
UINT DragQueryFile(hDrop, iFile, lpszFile, cb)

HDROP hDrop;         /* Handle einer internen
                        Datenstruktur                */
UINT iFile;          /* Index der abzufragenden Datei */
LPSTR lpszFile;      /* String für den Dateinamen     */
UINT cb;             /* Größe des Puffers für den
                        Dateinamen                   */
```

Beschreibung Die Funktion *DragQueryFile* ermittelt die Anzahl und die Namen abgelegter Dateien.

Parameter

hDrop: Bezeichnet die interne Datenstruktur, die die Dateinamen der abgelegten Dateien enthält. Dieses Handle wird im Parameter *wParam* der *WM_DROPFILES*-Botschaft an die Anwendung übergeben.

iFile: Gibt den Index der Datei an, die abgefragt werden soll. Der Index der ersten Datei ist 0. Wenn der Wert des Parameters *iFile* -1 ist, liefert *DragQueryFile* die Anzahl abgelegter Dateien zurück. Wenn der Wert des Parameters *iFile* zwischen Null und der Gesamtzahl abgelegter Dateien liegt, kopiert *DragQueryFile* den Dateinamen, der diesem Wert zugeordnet ist, in den Puffer, auf den der Parameter *lpszFile* zeigt.

lpszFile: Wenn die Funktion zurückkehrt, zeigt *lpszFile* auf einen nullterminierten String, der den Dateinamen einer abgelegten Datei enthält. Wenn dieser Parameter den Wert *NULL* hat und der Parameter *iFile* den Index des Namens der abgelegten Datei enthält, liefert *DropQueryFile* die erforderliche Größe des Puffers für diesen Dateinamen in Byte zurück.

cb: Gibt die Größe des Puffers *lpszFile* in Byte an.

Rückgabewert

Wenn die Funktion einen Dateinamen in den Puffer *lpszFile* kopiert, ist der Rückgabewert die Anzahl der kopierten Byte. Wenn der Parameter *iFile* 0xFFFF ist, ist der Rückgabwert die Anzahl der abgelegten Dateien. Ist *iFile* ein Wert zwischen Null und der Gesamtzahl der abgelegten Dateien und hat *lpszFile* den Wert *NULL*, ist der Rückgabewert die erforderliche Größe des Puffers *lpszFile*.

Siehe auch

DragQueryPoint

DragQueryPoint 3.1

Syntax

```
#include <shellapi.h>

BOOL DragQueryPoint(hDrop, lppt)
HDROP hDrop;            /* Handle der Struktur für
                           abgelegte Dateien          */
POINT FAR* lppt;        /* Adresse der Struktur für
                           Mauszeigerkoordinaten      */
```

Beschreibung

Die Funktion *DragQueryFile* ermittelt die Fensterkoordinaten des Mauszeigers, wenn eine Datei abgelegt wird.

hDrop: Bezeichnet die interne Datenstruktur, die die abgelegte Datei beschreibt. Diese Struktur wurde im Parameter *wParam* der *WM_DROPFILES*-Botschaft zurückgeliefert.

lppt: Zeigt auf eine *POINT*-Struktur, die die Funktion mit den Koordinaten der Position füllt, an der sich der Mauszeiger befand, als die Datei abgelegt wurde.

Die *POINT*-Struktur hat folgende Form:

```
typedef struct tagPOINT { /*  pt  */
   int x;
   int y;
} POINT;
```

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die Datei im Client-Bereich des Fensters abgelegt wird. Andernfalls ist er Null.

Hinweis Die Funktion *DragQueryPoint* füllt die *POINT*-Struktur mit den Koordinaten der Position, an der sich der Mauszeiger befand, als der Anwender die linke Maustaste losließ. Die Koordinaten werden für das Fenster zurückgeliefert, das die *WM_DROPFILES*-Botschaft erhalten hat.

Siehe auch *DragQueryFile*

DriverProc 3.1

Syntax

```
LRESULT CALLBACK DriverProc(dwDriverIdentifier,
   hDriver, wMessage, lParam1, lParam2)

DWORD hwDriverIdentifier;  /* Bezeichnet den instal-
                              lierbaren Treiber      */
HDRVR hDriver;             /* Handle des instal-
                              lierbaren Treibers     */
UINT wMessage;             /* Botschaft              */
LPARAM lParam1;            /* erster Botschafts-
                              parameter              */
LPARAM lParam2;            /* zweiter Botschafts-
                              parameter              */
```

Beschreibung Die Funktion *DriverProc* bearbeitet die angegebene Botschaft.

Parameter *dwDriverIdentifier:* Bestimmt den Bezeichner eines installierbaren Treibers.

hDriver: Bezeichnet den installierbaren Treiber. Dieser Parameter ist ein eindeutiges Handle, das Windows dem Treiber zuordnet.

wMessage: Bezeichnet eine Botschaft, die der Treiber bearbeiten muß. Folgende Botschaften können von Windows oder einer Anwendung an einen installierbaren Treiber übergeben werden:

Botschaft	Beschreibung
DRV_CLOSE	Benachrichtigt den Treiber, daß er seinen Verwendungszähler um eins vermindern soll. Ist der Zähler Null, wird der Treiber aus dem Speicher entfernt.

Botschaft	Beschreibung
DRV_CONFIGURE	Benachrichtigt den Treiber, daß er ein Dialogfenster zur Konfiguration darstellen soll. Diese Botschaft sollte nur übergeben werden, wenn der Treiber einen Wert ungleich Null zurückliefert, nachdem die Botschaft *DRV_QUERYCONFIGURE* bearbeitet wurde.
DRV_DISABLE	Benachrichtigt den Treiber, daß der von ihm belegte Speicher freigegeben werden soll.
DRV_ENABLE	Benachrichtigt den Treiber, daß er geladen oder erneut geladen wurde oder daß Windows aktiviert wurde.
DRV_FREE	Teilt dem Treiber mit, daß er verworfen wird.
DRV_INSTALL	Benachrichtigt den Treiber, daß er erfolgreich installiert wurde.
DRV_LOAD	Benachrichtigt den Treiber, daß er erfolgreich geladen wurde.
DRV_OPEN	Teilt dem Treiber mit, daß er geöffnet werden soll.
DRV_POWER	Teilt dem Treiber mit, daß die Stromversorgung des Geräts gerade an- oder ausgeschaltet wird.
DRV_QUERY-CONFIGURE	Ermittelt, ob der Treiber die Botschaft *DRV_CONFIGURE* unterstützt. Diese Botschaft führt zur Anzeige eines privaten Dialogfensters zur Konfiguration.
DRV_REMOVE	Teilt dem Treiber mit, daß er aus dem System entfernt werden soll.

lParam1: Bestimmt den ersten Botschaftsparameter.

lParam2: Bestimmt den zweiten Botschaftsparameter.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *DriverProc* ist die Hauptfunktion eines installierbaren Windows-Treibers. Sie wird vom Entwickler des Treibers bereitgestellt.

Wenn der Parameter *wMessage* den Wert *DRV_OPEN* hat, ist *lParam1* der String, der hinter dem Dateinamen des Treibers in der Datei SYSTEM.INI steht. *lParam2* ist der Wert, der als Parameter *lParam* im Aufruf der Funktion *OpenDriver* übergeben wird. Hat der Parameter *wMessage* den Wert *DRV_CLOSE*,

enthalten *lParam1* und *lParam2* die gleichen Werte wie die Parameter *lParam1* und *lParam2* im Aufruf der Funktion *CloseDriver*.

Siehe auch *CloseDriver, OpenDriver*

EnableCommNotification 3.1

Syntax

```
BOOL EnableCommNotification(idComDev, hwnd,
  cbWriteNotify, cbOutQueue)

int idComDev;      /* Bezeichner des seriellen Geräts */
HWND hwnd;         /* Handle des Fensters, das die
                      Botschaften empfängt           */
int cbWriteNotify; /* Anzahl der Byte, die vor der
                      Benachrichtigung geschrieben
                      wurden                         */
int cbOutQueue;    /* minimale Anzahl der Byte in der
                      Ausgabewarteschlange           */
```

Beschreibung Die Funktion *EnableCommNotification* aktiviert oder deaktiviert die Übergabe von *WM_COMMNOTIFY*-Botschaften an das angegebene Fenster.

Parameter *idComDev:* Bestimmt das Gerät, das Benachrichtigungsbotschaften an das durch *hwnd* bezeichnete Fenster übergibt. Die Funktion *OpenComm* gibt den Wert für diesen Parameter zurück.

hwnd: Bezeichnet das Fenster, für das die Botschaftsübergabe aktiviert oder deaktiviert wird. Ist *hWnd NULL*, deaktiviert *EnableCommNotification* die Botschaftsübergabe für das aktuelle Fenster.

cbWriteNotify: Bestimmt die Anzahl der Byte, die der COM-Gerätetreiber in die Eingabewarteschlange der Anwendung schreiben muß, bevor er eine Benachrichtigungsbotschaft übergibt. Die Botschaft weist die Anwendung an, die Informationen aus der Eingabewarteschlange zu lesen.

cbOutQueue: Bestimmt die kleinstmögliche Anzahl der Byte in der Ausgabewarteschlange. Fällt diese Anzahl unter den Wert von *cbOutQueue*, übergibt der COM-Gerätetreiber eine Benachrichtigungsbotschaft. Diese Botschaft weist die Anwendung an, Daten in die Ausgabewarteschlange zu schreiben.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich war. Der Rückgabewert Null zeigt an, daß eine ungültige ID für die COM-Schnittstelle verwendet wurde, die Schnittstelle nicht geöffnet ist oder daß die Funktion nicht von COMM.DRV unterstützt wird.

Hinweis

Verwendet eine Anwendung den Wert -1 für den Parameter *cbWriteNotify*, wird die Botschaft *WM_COMMNOTIFY* für die Benachrichtigungen *CN_EVENT* und *CN_TRANSMIT*, nicht aber für *CN_RECEIVE* an das angegebene Fenster übergeben. Wird -1 für den Parameter *cbOutQueue* angegeben, werden die Benachrichtigungen *CN_EVENT* und *CN_RECEIVE*, nicht aber *CN_TRANSMIT* übergeben.

Tritt ein Timeout auf, bevor die im Parameter *cbWriteNotify* angegebene Anzahl Byte in die Eingabewarteschlange geschrieben wurde, wird die Botschaft *WM_COMMNOTIFY* mit dem gesetzten Flag *CN_RECEIVE* übergeben. In diesem Fall wird keine weitere Botschaft übergeben, bis die Anzahl der Byte in der Eingabewarteschlange unter den Wert des Parameters *cbWriteNotify* fällt. Entsprechend wird eine *WM_COMMNOTIFY*-Botschaft mit gesetztem Flag *CN_RECEIVE* nur dann übergeben, wenn die Anzahl der Byte in der Ausgabewarteschlange größer ist als der im Parameter *cbOutQueue* angegebene Wert.

Die Version von COMM.DRV für Windows 3.0 unterstützt diese Funktion nicht.

EnableScrollBar 3.1

Syntax

```
BOOL EnableScrollBar(hwnd, fnSBFlags, fuArrowFlags)

HWND hwnd;              /* Handle des Fensters
                           oder der Bildlaufleiste     */
int fnSBFlags;          /* Typ-Flag der Bildlaufleiste */
UINT fuArrowFlags;      /* Pfeil-Flag der
                           Bildlaufleiste              */
```

Beschreibung

Die Funktion *EnableScrollBar* aktiviert oder deaktiviert einen oder beide Pfeile einer Bildlaufleiste.

Parameter

hwnd: Bezeichnet in Abhängigkeit von *fnSBFlags* ein Fenster oder eine Bildlaufleiste.

fnSBFlags: Bestimmt den Typ der Bildlaufleiste. Folgende Werte sind möglich:

Wert	Bedeutung
SB_BOTH	Aktiviert oder deaktiviert die Pfeile der horizontalen und der vertikalen Bildlaufleiste des angegebenen Fensters. Der Parameter *hwnd* bezeichnet das Fenster.
SB_CTL	Die Bildlaufleiste ist ein Steuerelement. Der Parameter *hwnd* muß ein Bildlaufleistensteuerelement bezeichnen.

Wert	Bedeutung
SB_HORZ	Aktiviert oder deaktiviert die Pfeile der horizontalen Bildlaufleiste des angegebenen Fensters. Der Parameter *hwnd* bezeichnet das Fenster.
SB_VERT	Aktiviert oder deaktiviert die Pfeile der vertikalen Bildlaufleiste des angegebenen Fensters. Der Parameter *hwnd* bezeichnet das Fenster.

fuArrowFlags: Bestimmt, ob und welche Pfeile einer Bildlaufleiste aktiviert oder deaktiviert werden. Folgende Werte sind möglich:

Wert	Bedeutung
ESB_ENABLE_BOTH	Aktiviert beide Pfeile einer Bildlaufleiste.
ESB_DISABLE_LTUP	Deaktiviert den linken Pfeil einer horizontalen bzw. den oberen Pfeil einer vertikalen Bildlaufleiste.
ESB_DISABLE_RTDN	Deaktiviert den rechten Pfeil einer horizontalen bzw. den unteren Pfeil einer vertikalen Bildlaufleiste.
ESB_DISABLE_BOTH	Deaktiviert beide Pfeile einer Bildlaufleiste.

Rückgabewert

Der Rückgabewert ist ungleich Null, wenn die Pfeile wie angegeben aktiviert bzw. deaktiviert wurden. Er ist Null, wenn sich die Pfeile bereits im angeforderten Zustand befinden oder ein Fehler auftritt.

Beispiel

Das folgende Beispiel aktiviert die vertikale Bildlaufleiste eines Editierfelds, wenn es den Fokus erhält. Die Bildlaufleiste wird deaktiviert, sobald das Editierfeld den Fokus verliert.

```
case EN_SETFOCUS:
    EnableScrollBar(hwndMLEdit, SB_VERT,
                    ESB_ENABLE_BOTH);
    break;

case EN_KILLFOCUS:
    EnableScrollBar(hwndMLEdit, SB_VERT,
                    ESB_DISABLE_BOTH);
    break;
```

Siehe auch

ShowScrollBar

EndDoc 3.1

Syntax

```
int EndDoc(hdc)

HDC hdc;              /* Handle des Gerätekontexts    */
```

Beschreibung Die Funktion *EndDoc* beendet einen Druckauftrag. Diese Funktion ersetzt den Drucker-Escape-Code *ENDDOC*.

Parameter *hdc:* Bezeichnet den Gerätekontext für den Druckauftrag.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert größer oder gleich Null, andernfalls kleiner als Null.

Hinweis Anwendungen sollten die Funktion *EndDoc* unmittelbar nach der erfolgreichen Beendigung eines Druckauftrags aufrufen. Um einen Druckauftrag zu beenden, wenn ein Fehler aufgetreten ist oder der Benutzer *Abbrechen* gewählt hat, sollten Anwendungen die Funktion *AbortDoc* aufrufen.

Verwenden Sie die Funktion *EndDoc* nicht in Metafiles.

Siehe auch *AbortDoc*, *Escape*, *StartDoc*

EndPage 3.1

Syntax

```
int EndPage(hdc)

HDC hdc;              /* Handle des Gerätekontexts    */
```

Beschreibung Die Funktion *EndPage* teilt einem Gerät mit, daß eine Anwendung das Beschreiben einer Seite beendet hat. Diese Funktion dient normalerweise dem Senden eines Seitenvorschubbefehls an den Drucker.

Diese Funktion ersetzt den Drucker-Escape-Code *NEWFRAME* ab Windows 3.1. Im Unterschied zu *NEWFRAME* wird diese Funktion immer nach dem Drucken einer Seite aufgerufen.

Parameter *hdc:* Bezeichnet den Gerätekontext für den Druckauftrag.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert größer oder gleich Null. Andernfalls handelt es sich um einen Fehlerwert.

Fehler Wenn die Funktion fehlschlägt, gibt sie einen der folgenden Fehlerwerte zurück:

Wert	Bedeutung
SP_ERROR	Allgemeiner Fehler.
SP_APPABORT	Der Druckauftrag wurde abgebrochen, weil die Abbruchfunktion der Anwendung Null zurückgeliefert hat.

Wert	Bedeutung
SP_USERABORT	Der Anwender hat den Auftrag mit Hilfe des Druck-Managers (PRINTMAN.EXE) von Windows abgebrochen.
SP_OUTOFDISK	Es steht nicht genug Plattenspeicherplatz für die Aufzeichnung des Druckauftrags zur Verfügung. Weiterer Platz kann nicht bereitgestellt werden.
SP_OUTOFMEMORY	Für die Aufzeichnung des Druckauftrags steht nicht genug Speicher zur Verfügung.

Hinweis Die Funktion *ResetDC* kann nötigenfalls nach dem Aufruf der Funktion *EndPage* aufgerufen werden, um den Modus des Geräts zu ändern.

Siehe auch *Escape*, *ResetDC*, *StartPage*

EnumFontFamilies 3.1

Syntax

```
int EnumFontFamilies(hdc, lpszFamily, fntenmprc,
   lParam)

HDC hdc;                /* Handle des Gerätekontexts */
LPCSTR lpszFamily;      /* Adresse des Namens der
                           Schriftfamilie           */
FONTENUMPROC fntenmprc; /* Adresse der
                           Callback-Funktion        */
LPARAM lParam;          /* anwendungsdefinierte
                           Daten                    */
```

Beschreibung Die Funktion *EnumFontFamilies* listet die Schriften der angegebenen Schriftfamilie auf, die auf einem bestimmten Gerät verfügbar sind. *EnumFontFamilies* führt die Auflistung fort, bis keine weiteren Schriften vorhanden sind oder die Callback-Funktion Null zurückgibt.

Parameter *hdc:* Bezeichnet den Gerätekontext.

lpszFamily: Zeigt auf einen nullterminierten String, der den Familiennamen der gewünschten Schrift angibt. Wenn dieser Parameter *NULL* ist, wählt und listet die Funktion *EnumFontFamilies* eine Schrift aus jeder verfügbaren Schriftfamilie.

fntenmprc: Bestimmt die Adresse der Prozedurinstanz der anwendungsdefinierten Callback-Funktion. Die Adresse muß mit der Funktion *MakeProcInstance* erzeugt werden. Weitere Informationen über die Callback-Funktion finden Sie in der Beschreibung der Callback-Funktion *EnumFontFamProc*.

lParam: Bestimmt einen anwendungsdefinierten 32-Bit-Wert, der zusammen mit Schriftinformationen an die Callback-Funktion übergeben wird.

Rückgabewert Der Rückgabewert ist der letzte von der Callback-Funktion zurückgegebene Wert. Dieser Wert hängt davon ab, welche Schriftfamilien für das angegebene Gerät verfügbar sind.

Hinweis Die Funktion *EnumFontFamilies* unterscheidet sich von der Funktion *EnumFonts*, da sie auch die Schriftarten einer TrueType-Schrift abruft. Mit *EnumFontFamilies* können Anwendungen Informationen über ungewöhnliche Schriftarten (z. B. Outline) abrufen, die von der Funktion *EnumFonts* nicht aufgelistet werden können. Anwendungen sollten die Funktion *EnumFontFamilies* anstelle von *EnumFonts* benutzen.

Die Funktion *EnumFontFamilies* ruft Informationen über jede Schrift ab, die den durch den Parameter *lpszFamily* bezeichneten Schriftnamen hat und übergibt sie an die Funktion, die durch den Parameter *fntenmprc* bezeichnet wird. Diese von der Anwendung bereitgestellte Callback-Funktion kann die Schriftinformationen wie gewünscht bearbeiten.

Beispiel Das folgende Beispiel verwendet die Funktion *MakeProcInstance*, um einen Zeiger auf die Callback-Funktion für die Funktion *EnumFontFamilies* zu erzeugen. Die Funktion *FreeProcInstance* wird aufgerufen, wenn die Auflistung beendet ist. Da der zweite Parameter *NULL* ist, listet *EnumFontFamilies* eine Schrift aus jeder Familie auf, die im angegebenen Gerätekontext verfügbar ist. Die Variable *aFontCount* zeigt auf ein Array, das innerhalb der Callback-Funktion benutzt wird.

```
FONTENUMPROC lpEnumFamCallBack;
int aFontCount[] = { 0, 0, 0 };

lpEnumFamCallBack = (FONTENUMPROC)
    MakeProcInstance((FARPROC) EnumFamCallBack,
                     hAppInstance);
EnumFontFamilies(hdc, NULL, lpEnumFamCallBack,
                 (LPARAM) aFontCount);
FreeProcInstance((FARPROC) lpEnumFamCallBack);
```

Siehe auch *EnumFonts, EnumFontFamProc*

EnumFontFamProc 3.1

Syntax

```
int CALLBACK EnumFontFamProc(lpnlf, lpntm, FontType,
    lParam)

LOGFONT FAR* lpnlf;    /* Adresse der Struktur mit den
                          virtuellen Schriftdaten     */
```

```
TEXTMETRIC FAR* lpntm; /* Adresse der Struktur mit den
                          physikalischen Schriftdaten */
int FontType;          /* Schrifttyp                  */
LPARAM lParam;         /* Adresse der anwendungs-
                          definierten Daten           */
```

Beschreibung

Die Funktion *EnumFontFamProc* ist eine anwendungsdefinierte Callback-Funktion, die Informationen über verfügbare Schriften abruft.

Parameter

lpnlf: Zeigt auf eine *NEWLOGFONT*-Struktur, die Informationen über die virtuellen Attribute der Schrift enthält. Diese Struktur ist lokal definiert und entspricht bis auf zwei Elemente der Windows-Struktur *LOGFONT*.

Die *NEWLOGFONT*-Struktur hat folgende Form:

```
typedef struct tagNEWLOGFONT {       /* nlf */
    int lfHeight;
    int lfWidth;
    int lfEscapement;
    int lfOrientation;
    int lfWeight;
    BYTE lfItalic;
    BYTE lfUnderline;
    BYTE lfStrikeOut;
    BYTE lfCharSet;
    BYTE lfOutPrecision;
    BYTE lfClipPrecision;
    BYTE lfQuality;
    BYTE lfPitchAndFamily;
    BYTE lfFaceName[LF_FACESIZE];
    BYTE lfFullName[2 * LF_FACESIZE]; /* nur TrueType */
    BYTE  lfStyle[LF_FACESIZE];       /* nur TrueType */
}
NEWLOGFONT;
```

Die Elemente *lfFullName* und *lfStyle* werden einer *LOGFONT*-Struktur hinzugefügt, wenn eine TrueType-Schrift mit der Funktion *EnumFontFamProc* aufgelistet wird.

Das Element *lfFullName* ist ein Zeichen-Array, das den vollständigen Namen dieser Schrift angibt. Dieser Name enthält den Schriftnamen und den Auszeichnungsnamen.

Das Element *lfStyle* ist ein Zeichen-Array, das den Auszeichnungsnamen der Schrift angibt.

Wenn zum Beispiel die Schrift Arial® mit den Auszeichnungen fett und kursiv aufgelistet wird, enthalten die letzten drei Elemente der *NEWLOGFONT*-Struktur folgende Strings:

```
lfFaceName = "Arial";
lfFullName = "Arial Bold Italic";
lfStyle = "Bold Italic";
```

lpntm: Zeigt auf eine *NEWTEXTMETRIC*-Struktur, die Informationen über die physikalischen Attribute der Schrift enthält, wenn es sich um eine TrueType-Schrift handelt. Liegt keine TrueType-Schrift vor, zeigt dieser Parameter auf eine *TEXTMETRIC*-Struktur.

Die *NEWTEXTMETRIC*-Struktur hat folgende Form:

```
typedef struct tagNEWTEXTMETRIC {   /* ntm */
    int tmHeight;
    int tmAscent;
    int tmDescent;
    int tmInternalLeading;
    int tmExternalLeading;
    int tmAveCharWidth;
    int tmMaxCharWidth;
    int tmWeight;
    BYTE tmItalic;
    BYTE tmUnderlinded;
    BYTE tmStruckOut;
    BYTE tmFirstChar;
    BYTE tmLastChar;
    BYTE tmDefaultChar;
    BYTE tmBreakChar;
    BYTE tmPitchAndFamily;
    BYTE tmCharSet;
    int tmOverhang;
    int tmDigitizedAspectX;
    int tmDigitizedAspectY;
    DWORD ntmFlags;
    UINT ntmSizeEM;
    UINT ntmCellHeight;
    UINT ntmAvgWidth;
} NEWTEXTMETRIC;
```

Die *TEXTMETRIC*-Struktur ist mit der *NEWTEXTMETRIC*-Struktur weitgehend identisch. Ihr fehlen jedoch die letzten vier Elemente.

FontType: Bestimmt den Typ der Schrift. Dieser Parameter kann eine Kombination der folgenden Bit-Masken sein:

DEVICE_FONTTYPE
RASTER_FONTTYPE
TRUETYPE_FONTTYPE

lParam: Zeigt auf die anwendungsdefinierten Daten, die von *EnumFontFamilies* übergeben werden.

Rückgabewert Damit die Auflistung fortgesetzt wird, muß diese Funktion einen Wert ungleich Null zurückggeben. Der Rückgabewert Null beendet die Auflistung.

Hinweis

Anwendungen müssen diese Callback-Funktion registrieren, indem sie ihre Adresse an die Funktion *EnumFontFamilies* übergeben. Der Name *EnumFontFamProc* ist ein Platzhalter für den anwendungsdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei (.DEF) der Anwendung exportiert werden.

Der Operator UND (&) kann mit den Konstanten *RASTER_FONTTYPE*, *DEVICE_FONTTYPE* und *TRUETYPE_FONTTYPE* benutzt werden, um den Typ der Schrift zu ermitteln. Ist das Bit *RASTER_FONTTYPE* gesetzt, handelt es sich um eine Rasterschrift. Ist das Bit *TRUETYPE_FONTTYPE* gesetzt, handelt es sich um eine TrueType-Schrift. Ist keines dieser Bits gesetzt, liegt eine Vektorschrift vor.

Das dritte Flag *DEVICE_FONTTYPE* wird gesetzt, wenn ein Gerät (zum Beispiel ein Laserdrucker) das Laden von TrueType-Schriften unterstützt. Es ist Null, wenn die Schrift keine Geräteschrift ist. Jedes Gerät kann Geräteschriften unterstützen, auch Grafikkarten und Nadeldrucker. Anwendungen können das Flag *DEVICE_FONTTYPE* verwenden, um zwischen GDI-unterstützten Rasterschriften und geräteunterstützten Schriften zu unterscheiden. Die GDI kann die Auszeichnungen Fett, Kursiv, Unterstrichen und Durchgestrichen für GDI-unterstützte Rasterschriften emulieren, nicht aber für geräteunterstützte Schriften.

Siehe auch *EnumFontFamilies, EnumFonts*

EnumFontsProc 3.1

Syntax

```
int CALLBACK EnumFontsProc(lplf, lpntm, FontType,
    lpData)

LOGFONT FAR* lplf;          /* Adresse der Struktur
                               für die virtuellen
                               Schriftdaten          */
NEWTEXTMETRIC FAR* lpntm;   /* Adresse der Struktur
                               für die physikalischen
                               Schriftdaten          */
int FontType;               /* Typ der Schrift       */
LPARAM lpData;              /* Adresse der anwendungs-
                               definierten Daten     */
```

Beschreibung

Die Funktion *EnumFontsProc* ist eine anwendungsdefinierte Callback-Funktion, die Schriftdaten der Funktion *EnumFonts* bearbeitet.

Parameter

lplf: Zeigt auf eine *LOGFONT*-Struktur, die Informationen über die virtuellen Attribute der Schrift enthält. Die *LOGFONT*-Struktur hat folgende Form:

```
typedef struct tagLOGFONT {   /* lf */
    int lfHeight;
    int lfWidth;
    int lfEscapement;
    int lfOrientation;
    int lfWeight;
    BYTE lfItalic;
    BYTE lfUnderline;
    BYTE lfStrikeOut;
    BYTE lfCharSet;
    BYTE lfOutPrecision;
    BYTE lfClipPrecision;
    BYTE lfQuality;
    BYTE lfPitchAndFamily;
    BYTE lfFaceName[LF_FACESIZE];
} LOGFONT;
```

lpntm: Zeigt auf eine *NEWTEXTMETRIC*-Struktur, die Informationen über die physikalischen Attribute der Schrift enthält, wenn es sich um eine TrueType-Schrift handelt. Liegt keine TrueType-Schrift vor, zeigt *lpntm* auf eine *TEXTMETRIC*-Struktur. Die *NEWTEXTMETRIC*-Struktur hat folgende Form:

```
typedef struct tagNEWTEXTMETRIC {   /* ntm */
    int tmHeight;
    int tmAscent;
    int tmDescent;
    int tmInternalLeading;
    int tmExternalLeading;
    int tmAveCharWidth;
    int tmMaxCharWidth;
    int tmWeight;
    BYTE tmItalic;
    BYTE tmUnderlinded;
    BYTE tmStruckOut;
    BYTE tmFirstChar;
    BYTE tmLastChar;
    BYTE tmDefaultChar;
    BYTE tmBreakChar;
    BYTE tmPitchAndFamily;
    BYTE tmCharSet;
    int tmOverhang;
    int tmDigitizedAspectX;
    int tmDigitizedAspectY;
    DWORD ntmFlags;
    UINT ntmSizeEM;
    UINT ntmCellHeight;
    UINT ntmAvgWidth;
} NEWTEXTMETRIC;
```

Die *TEXTMETRIC*-Struktur ist fast identisch mit der *NEWTEXTMETRIC*-Struktur. Ihr fehlen nur die letzten 4 Elemente.

FontType: Bestimmt den Typ der Schrift. Dieser Parameter kann eine Kombination der folgenden Bit-Masken sein:

DEVICE_FONTTYPE
RASTER_FONTTYPE
TRUETYPE_FONTTYPE

lpData: Zeigt auf die anwendungsdefinierten Daten, die von der Funktion *EnumFonts* übergeben werden.

Rückgabewert Damit die Auflistung fortgesetzt wird, muß diese Funktion einen Wert ungleich Null zurückgeben. Der Rückgabewert Null beendet die Auflistung.

Hinweis Anwendungen müssen diese Callback-Funktion registrieren, indem sie ihre Adresse an die Funktion *EnumFonts* übergeben. Der Name *EnumFontsProc* ist ein Platzhalter für den anwendungsdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei (.DEF) der Anwendung exportiert werden.

Der Operator UND (&) kann mit den Konstanten *RASTER_FONTTYPE*, *DEVICE_FONTTYPE* und *TRUETYPE_FONTTYPE* benutzt werden, um den Typ der Schrift zu ermitteln. Ist das Bit *RASTER_FONTTYPE* gesetzt, handelt es sich um eine Rasterschrift. Ist das Bit *TRUETYPE_FONTTYPE* gesetzt, handelt es sich um eine TrueType-Schrift. Ist keines dieser Bits gesetzt, liegt eine Vektorschrift vor. Das dritte Flag *DEVICE_FONTTYPE* wird gesetzt, wenn ein Gerät (zum Beispiel ein Laserdrucker) das Laden von TrueType-Schriften unterstützt. Es ist Null, wenn die Schrift keine Geräteschrift ist. Jedes Gerät kann Geräteschriften unterstützen, auch Graphikkarten und Nadeldrucker. Anwendungen können das Flag *DEVICE_FONTTYPE* verwenden, um zwischen GDI-unterstützten Rasterschriften und geräteunterstützten Schriften zu unterscheiden. Die GDI kann die Auszeichnungen Fett, Kursiv, Unterstrichen und Durchgestrichen für GDI-unterstützte Rasterschriften emulieren, nicht aber für geräteunterstützte Schriften.

Siehe auch *EnumFonts, EnumFontsFamilies*

EnumMetaFileProc 3.1

Syntax

```
int CALLBACK EnumMetaFileProc(hdc, lpht, lpmr, cObj,
    lParam)

HDC hdc;                /* Handle des Gerätekontexts */
HANDLETABLE FAR* lpht;  /* Adresse der Tabelle
                           der Objekt-Handles       */
```

```
METARECORD FAR* lpmr;  /* Adresse des Metafile-Record */
int cObj;              /* Anzahl der Objekte in
                          der Handle-Tabelle          */
LPARAM lParam;         /* Adresse der anwendungs-
                          definierten Daten           */
```

Beschreibung Die Funktion *EnumMetaFileProc* ist eine anwendungsdefinierte Callback-Funktion, die Metafile-Daten der Funktion *EnumMetaFile* bearbeitet.

Parameter *hdc:* Bezeichnet den Gerätekontext, der das Metafile enthält.

lpht: Zeigt auf die Tabelle der den Objekten (Pinsel, Stifte usw.) zugeordneten Handles. Diese Tabelle bezieht sich auf Objekte im Metafile.

lpmr: Zeigt auf ein Metafile-Record im Metafile.

cObj: Bestimmt die Anzahl der Objekte mit zugeordnetem Handle in der Handle-Tabelle.

lParam: Zeigt auf die anwendungsdefinierten Daten.

Rückgabewert Die Callback-Funktion muß einen Wert ungleich Null zurückgeben, damit die Auflistung fortgesetzt wird. Der Rückgabewert Null beendet die Auflistung.

Hinweis Anwendungen müssen diese Callback-Funktion registrieren, indem sie ihre Adresse an die Funktion *EnumMetaFile* übergeben.

Der Name *EnumMetaFileProc* ist ein Platzhalter für den anwendungsdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei (.DEF) der Anwendung exportiert werden.

Siehe auch *EnumMetaFile*

EnumObjectsProc 3.1

Syntax

```
int CALLBACK EnumObjectsProc(lpLogObject, lpData)

void FAR* lpLogObject;   /* Adresse des Objekts       */
LPARAM lpData;           /* Adresse der anwendungs-
                            definierten Daten         */
```

Beschreibung *EnumObjectsProc* ist eine anwendungsdefinierte Callback-Funktion, die Objektdaten der Funktion *EnumObjects* bearbeitet.

Parameter *lpLogObject:* Zeigt auf eine *LOGPEN*- oder *LOGBRUSH*-Struktur, die Informationen über die Attribute des Objekts enthält.

Die *LOGPEN*-Struktur hat folgende Form:

```
typedef struct tagLOGPEN {   /* lgpn */
    UINT       lopnStyle;
```

```
    POINT      lopnWidth;
    COLORBREF  lopnColor;
} LOGPEN;
```

Die *LOGBRUSH*-Struktur hat folgende Form:

```
typedef struct tagLOGBRUSH {   /* lb */
    UINT       lbStyle;
    COLORBREF  lbColor;
    int        lbHatch;
} LOGBRUSH;
```

lpData: Zeigt auf die anwendungsdefinierten Daten, die von der Funktion *EnumObjects* übergeben werden.

Rückgabewert

Diese Funktion muß einen Wert ungleich Null zurückgeben, damit die Auflistung fortgesetzt wird. Der Rückgabewert Null beendet die Auflistung.

Hinweis

Anwendungen müssen diese Callback-Funktion registrieren, indem sie ihre Adresse an die Funktion *EnumObjects* übergeben. Der Name *EnumObjectsProc* ist ein Platzhalter für den anwendungsdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei (.DEF) der Anwendung exportiert werden.

Beispiel

Das folgende Beispiel ruft die Anzahl der horizontal schraffierten Pinsel ab und füllt *LOGBRUSH*-Strukturen mit Informationen über jeden einzelnen:

```
#define MAXBRUSHES 50

GOBJENUNMPROC lpProcCallback;
HGLOBAL hglbl;
LPBYTE lpbCountBrush;

lpProcCallback = (GOBJENUMPROC) MakeProcInstance(
    (FARPROC) Callback, hinst);

hglbl = GlobalAlloc(GMEM_FIXED, sizeof(LOGBRUSH)
    * MAXBRUSHES);
lpbCountBrush = (LPBYTE) GlobalLock(hglbl);
*lpbCountBrush = 0;
EnumObjects(hdc, OBJ_BRUSH, lpProcCallback,
    (LPARAM) lpbCountBrush);

FreeProcInstance((FARPROC) lpProcCallback);

int FAR PASCAL Callback(LPLOGBRUSH lpLogBrush,
                        LPBYTE pbData)
{
    /* Der Parameter pbData enthält die Anzahl der
       horizontal schraffierten Pinsel. Der Parameter
       lpDest wird mit dem Wert initialisiert, der dem
```

```
        für pbData reservierten Byte und den
        LOGBRUSH-Strukturen folgt, die mit Informationen
        über die Pinsel gefüllt wurden. */

    LPLOGBRUSH lpDest =
        (LPLOGBRUSH) (pbData + 1 +
        (*pbData * sizeof(LOGBRUSH)));
        if (lpLogBrush->lbStyle ==
            BS_HATCHED &&        /* wenn horizontale
                                    Schraffur vorliegt */
             lpLogBrush->lbHatch == HS_HORIZONTAL) {
        *lpDest++ = *lpLogBrush; /* füllt die Struktur
                                    mit Informationen
                                    über den Pinsel */
        (*pbData) ++;            /* erhöht den Zähler
                                    der Pinsel      */
        if (*pbData >= MAXBRUSHES)
            return 0;
      }
      return 1;
    }
```

Siehe auch *EnumObjects, FreeProcInstance, GlobalAlloc, GlobalLock, MakeProcInstance*

EnumPropFixedProc 2.x

Syntax

```
BOOL CALLBACK EnumPropFixedProc(hwnd, lpsz, hData)

HWND hwnd;          /* Handle des Fensters mit der
                       Eigenschaft                  */
LPCSTR lpsz;        /* Adresse des Eigenschafts-
                       String oder -atoms           */
HANDLE hData;       /* Handle der Eigenschaftsdaten */
```

Beschreibung Die Funktion *EnumPropFixedProc* ist eine anwendungsdefinierte Callback-Funktion, die die Eigenschaftsdaten eines Fensters als Ergebnis eines Aufrufs der Funktion *EnumProps* erhält.

Parameter *hwnd:* Bezeichnet das Handle des Fensters, das die Eigenschaftsliste enthält.

lpsz: Zeigt auf den nullterminierten String, der den durch den Parameter *hData* bezeichneten Eigenschaftsdaten zugeordnet ist. Der String und die Daten wurden von der Anwendung in einem vorhergehenden Aufruf der Funktion *SetProp* festgelegt. Übergibt die Anwendung ein Atom anstelle eines String an *SetProp*, enthält der Parameter *lpsz* das Atom im niederwertigen und Null im höherwertigen *WORD.*

hData: Bezeichnet die Eigenschaftsdaten.

Rückgabewert Die Callback-Funktion muß *TRUE* zurückgeben, damit die Auflistung fortgesetzt wird. *FALSE* beendet die Auflistung.

Hinweis Diese Form einer Callback-Funktion für die Auflistung von Eigenschaften sollte in Anwendungen und dynamischen Linkbibliotheken mit festen Datensegmenten sowie in dynamischen Linkbibliotheken mit verschiebbaren Datensegmenten, die aber keinen Stack enthalten, benutzt werden. Folgende Einschränkungen gelten für die Callback-Funktion:

- Die Callback-Funktion darf die Steuerung weder abgeben noch eine Aktion ausführen, die zur Übergabe der Steuerung an andere Tasks führen könnte.
- Die Callback-Funktion kann die Funktion *RemoveProp* aufrufen. Die Funktion *RemoveProp* kann jedoch nur die Eigenschaften entfernen, die über einen Parameter an die Callback-Funktion übergeben wurden.
- Die Callback-Funktion darf keine Eigenschaften hinzufügen.

Der Name *EnumPropFixedProp* ist ein Platzhalter für den anwendungsdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei (.DEF) der Anwendung exportiert werden.

Siehe auch *EnumPropMovableProc*, *EnumProps*, *RemoveProp*, *SetProp*

EnumPropMovableProc 2.x

Syntax

```
BOOL CALLBACK EnumPropMovableProc(hwnd, lpsz, hData)

HWND hwnd;          /* Handle des Fensters mit der
                       Eigenschaft                 */
LPCSTR lpsz;        /* Adresse des Eigenschafts-
                       String oder -atoms          */
HANDLE hData;       /* Handle der Eigenschaftsdaten */
```

Beschreibung Die Funktion *EnumPropMovableProc* ist eine anwendungsdefinierte Callback-Funktion, die die Eigenschaftsdaten eines Fensters als Ergebnis eines Aufrufs der Funktion *EnumProps* erhält.

Parameter *hwnd:* Bezeichnet das Handle des Fensters, das die Eigenschaftsliste enthält.

lpsz: Zeigt auf den nullterminierten String, der den durch den Paramter *hData* bezeichneten Daten zugeordnet ist. Die Daten und der String wurden von der Anwendung in einem vorhergehenden Aufruf der Funktion *SetProp* angegeben. Wenn die Anwendung ein Atom anstelle eines String an *SetProp* übergibt, enthält der Parameter *lpsz* das Atom.

hData: Bezeichnet die Eigenschaftsdaten.

Rückgabewert Die Callback-Funktion muß *TRUE* zurückgeben, um die Auflistung fortzusetzen. Der Rückgabewert *FALSE* beendet die Auflistung.

Hinweis Diese Form einer Callback-Funktion für die Auflistung von Eigenschaften sollte in Anwendungen benutzt werden, die über verschiebbare Datensegmente verfügen und in dynamischen Linkbibliotheken, deren verschiebbare Datensegmente auch einen Stack enthalten. Diese Form wird benötigt, da die Verschiebung der Daten alle **long**-Zeiger auf Stack-Variablen ungültig macht, also auch den Parameter *lpsz*. Das Datensegment wird normalerweise dann verschoben, wenn die Callback-Funktion mehr Platz im lokalen Heap reserviert, als aktuell verfügbar ist.

Folgende Einschränkungen gelten für diese Callback-Funktion:

- Die Callback-Funktion darf die Steuerung nicht abgeben oder eine Funktion ausführen, die zur Übergabe der Steuerung an andere Tasks führen könnte.
- Die Callback-Funktion kann die Funktion *RemoveProp* aufrufen. *RemoveProp* kann jedoch nur die Eigenschaften entfernen, die über einen Parameter an die Callback-Funktion übergeben wurden.
- Die Callback-Funktion sollte keine Eigenschaften hinzufügen.

Der Name *EnumPropMovableProc* ist ein Platzhalter für den anwendungsdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei (.DEF) der Anwendung exportiert werden.

Siehe auch *EnumPropFixedProc*, *EnumProps*, *RemoveProp*, *SetProp*

EnumTaskWndProc 2.x

Syntax

```
BOOL CALLBACK EnumTaskWndProc(hwnd, lParam)

HWND hwnd;          /* Handle eines Fensters       */
LPARAM lParam;      /* anwendungsdefinierter Wert  */
```

Beschreibung Die Funktion *EnumTaskWndProc* ist eine anwendungsdefinierte Callback-Funktion, die das einer Task zugeordnete Fenster-Handle als Ergebnis eines Aufrufs der Funktion *EnumTaskWindows* erhält.

Parameter *hwnd:* Bezeichnet ein Fenster, das der in der Funktion *EnumTaskWindows* angegebenen Task zugeordnet ist.

lParam: Bestimmt den anwendungsdefinierten Wert, der in der Funktion *EnumTaskWindows* angegeben wurde.

Rückgabewert Die Callback-Funktion muß *TRUE* zurückgeben, um die Auflistung fortzusetzen. Der Rückgabewert *FALSE* beendet die Auflistung.

Hinweis Die Callback-Funktion kann jede gewünschte Aufgabe ausführen. Der Name *EnumTaskWndProc* ist ein Platzhalter für den anwendungsdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei (.DEF) der Anwendung exportiert werden.

Siehe auch *EnumTaskWindows*

EnumWindowsProc 2.x

Syntax

```
BOOL CALLBACK EnumWindowsProc(hwnd, lParam)

HWND hwnd;          /* Handle des übergeordneten
                       Fensters                  */
LPARAM lParam;      /* anwendungsdefinierter Wert */
```

Beschreibung Die Funktion *EnumWindowsProc* ist eine anwendungsdefinierte Callback-Funktion, die die Handles übergeordneter Fenster als Ergebnis eines Aufrufs der Funktion *EnumWindows* erhält.

Parameter *hwnd:* Bezeichnet ein übergeordnetes Fenster.

lParam: Bestimmt den anwendungsdefinierten Wert, der in der Funktion *EnumWindows* angegeben wurde.

Rückgabewert Die Callback-Funktion muß einen Wert ungleich Null zurückgeben, damit die Auflistung fortgesetzt wird. Der Rückgabewert Null beendet die Auflistung.

Hinweis Die Callback-Funktion kann jede gewünschte Aufgabe durchführen.

Der Name *EnumWindowsProc* ist ein Platzhalter für den anwendungsdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei (.DEF) der Anwendung exportiert werden.

Siehe auch *EnumWindows*

ExitWindowsExec 3.0

Syntax

```
BOOL ExitWindowsExec(lpszExe, lpszParams)

LPCSTR lpszExe;     /* Zeiger auf Pfadname          */
LPCSTR lpszParams;  /* Zeiger auf Kommandozeilen-
                       optionen                     */
```

Beschreibung Die Funktion *ExitWindowsExec* beendet Windows, führt eine angegebene DOS-Anwendung aus und startet Windows dann erneut.

Parameter *lpszExe:* Zeigt auf einen nullterminierten String, der den Pfad- und Dateinamen einer ausführbaren Datei angibt, die das System starten soll, nachdem Windows beendet wurde. Dieser String darf einschließlich des nullterminierenden Zeichens nicht länger als 128 Byte sein.

lpszParams: Zeigt auf einen nullterminierten String, der Parameter für die ausführbare Datei angibt, die vom Parameter *lpszExe* bestimmt wird. Dieser String darf einschließlich des nullterminierenden Zeichens nicht länger als 127 Byte sein. Dieser Wert kann *NULL* sein.

Rückgabewert Wenn die Funktion fehlschlägt, ist der Rückgabewert *FALSE*. Der Grund des Fehlers kann in einem Speicherzuweisungsfehler oder darin bestehen, daß eine der Anwendungen im System nicht beendet werden konnte.

Hinweis Die Funktion *ExitWindowsExec* wird normalerweise von Installationsprogrammen verwendet, um Teile von Windows zu ersetzen, die aktiv sind, während Windows läuft.

Siehe auch *ExitWindows*

ExtractIcon 3.1

Syntax

```
#include <shellapi.h>
HICON ExtractIcon(hinst, lpszExeName, iIcon)

HINSTANCE hinst;      /* Instanzen-Handle            */
LPCSTR lpszExeName;   /* Adresse des String für
                         den Dateinamen              */
UINT iIcon;           /* Index des abzurufenden
                         Symbols                     */
```

Beschreibung Die Funktion *ExtractIcon* ruft das Handle eines Symbols aus einer angegebenen ausführbaren Datei, dynamischen Linkbibliothek (DLL) oder aus einer Symboldatei ab.

Parameter *hinst:* Bezeichnet die Instanz der Anwendung, die die Funktion aufruft.

lpszExeName: Zeigt auf einen nullterminierten String, der den Namen einer ausführbaren Datei, einer dynamischen Linkbibliothek oder einer Symboldatei angibt.

iIcon: Bestimmt den Index des abzurufenden Symbols. Wenn dieser Parameter Null ist, gibt die Funktion das Handle des ersten Symbols in der angegebenen Datei zurück. Ist dieser Parameter -1, gibt die Funktion die Anzahl der Symbole in der angegebenen Datei zurück.

Rückgabewert Der Rückgabewert ist das Handle eines Symbols, wenn die Funktion erfolgreich durchgeführt wurde. Er ist 1, wenn die

durch den Parameter *lpszExeName* bestimmte Datei keine ausführbare Datei, dynamische Linkbibliothek oder Symboldatei ist. In allen anderen Fällen ist der Rückgabewert *NULL*. Dies zeigt an, daß die Datei keine Symbole enthält.

FindExecutable 3.1

Syntax

```
#include <shellapi.h>
HINSTANCE FindExecutable(lpszFile, lpszDir, lpszResult)

LPCSTR lpszFile;      /* Adresse des String für
                         den Dateinamen              */
LPCSTR lpszDir;       /* Adresse des String für
                         das Standardverzeichnis     */
LPSTR lpszResult;     /* Adresse des String für
                         den Namen der ausführbaren
                         Datei bei der Rückgabe      */
```

Beschreibung *FindExecutable* ruft den Namen einer ausführbaren Datei ab, die einem angegebenen Dateinamen zugeordnet ist.

Parameter *lpszFile:* Zeigt auf einen nullterminierten String, der den Dateinamen angibt. Dabei kann es sich um ein Dokument oder um eine ausführbare Datei handeln.

lpszDir: Zeigt auf einen nullterminierten String, der den Laufwerksbuchstaben und den Pfad für das Standardverzeichnis angibt.

lpszResult: Zeigt auf einen Puffer, der den Namen einer ausführbaren Datei enthält, wenn die Funktion zurückkehrt. Dieser nullterminierte String bestimmt die Anwendung, die gestartet wird, wenn der Befehl *Öffnen* aus dem Menü *Datei* im Datei-Manager gewählt wird.

Rückgabewert Der Rückgabewert ist größer als 32, wenn die Funktion erfolgreich ist. Ein Wert kleiner oder gleich 32 zeigt einen Fehler an.

Fehler Die Funktion *FindExecutable* gibt 31 zurück, wenn keine Zuordnung für den angegeben Dateityp existiert. Nachstehend sehen Sie die anderen möglichen Fehlerwerte:

Wert	Bedeutung
0	Nicht genügend Speicher im System, ausführbare Datei war beschädigt oder Relozierungen waren ungültig.
2	Datei nicht gefunden.
3	Pfad nicht gefunden.
5	Ein dynamischer Zugriff auf eine Task wurde versucht, oder es liegt ein Netzfehler vor.

Wert	Bedeutung
6	Die Bibliothek benötigt getrennte Datensegmente für jede Task.
8	Es steht nicht genügend Speicher zum Starten der Anwendung zur Verfügung.
10	Die Windows-Version ist falsch.
11	Die ausführbare Datei war ungültig. Entweder handelt es sich nicht um eine Windows-Anwendung oder der ausführbare Code weist einen Fehler auf.
12	Die Anwendung wurde für ein anderes Betriebssystem erstellt.
13	Die Anwendung wurde für MS-DOS 4.0 erstellt.
14	Der Typ der ausführbaren Datei ist unbekannt.
15	Es wurde versucht, eine Real-Modus-Anwendung zu laden, die für eine frühere Version von Windows entwickelt wurde.
16	Es wurde versucht, eine zweite Instanz einer ausführbaren Datei zu laden, die mehrere Datensegmente enthält, die nicht das Attribut Nur-Lesen besitzen.
19	Es wurde versucht, eine komprimierte ausführbare Datei zu laden. Die Datei muß entkomprimiert werden, bevor sie geladen werden kann.
20	Die dynamische Linkbibliothek (DLL) ist ungültig. Eine der von dieser Anwendung benötigten dynamischen Linkbibliotheken ist beschädigt.
21	Die Anwendung benötigt die 32-Bit-Erweiterungen von Microsoft Windows.

Hinweis

Der im Parameter *lpszFile* angegebene Dateiname ist mit einer ausführbaren Datei verknüpft, wenn eine Verknüpfung zwischen der Namenserweiterung der Datei und einer ausführbaren Datei in der Registrierungsdatenbank registriert worden ist. Anwendungen, die normalerweise eine bestimmte Namenserweiterung verwenden, verknüpfen diese Erweiterung meist mit der ausführbaren Datei, wenn sie installiert werden.

Siehe auch

RegQueryValue, *ShellExecute*

FindText 3.1

Syntax

```
#include <commdlg.h>
HWND FindText(lpfr);
```

```
FINDREPLACE FAR* lpfr;   /* Adresse der Struktur mit
                            den Initialisierungsdaten */
```

Beschreibung Die Funktion *FindText* erzeugt ein systemdefiniertes, nichtmodales Dialogfenster, mit dem der Anwender Textausschnitte in einem Dokument suchen kann. Die Anwendung muß die eigentliche Suchoperation ausführen.

Parameter *lpfr:* Zeigt auf eine *FINDREPLACE*-Struktur, die Initialisierungsinformationen für das Dialogenster enthält. Wenn der Anwender in diesem Dialogfenster eine Auswahl trifft, füllt das System diese Struktur mit den Informationen der Auswahl und sendet der Anwendung eine Botschaft. Diese Botschaft enthält einen Zeiger auf die *FINDREPLACE*-Struktur.

Die *FINDREPLACE*-Struktur hat folgende Form:

```
#include <commdlg.h>

typedef struct tagFINDREPLACE {   /* fr */
    DWORD lStructSize;
    HWND hwndOwner;
    HINSTANCE hInstance;
    DWORD Flags;
    LPSTR lpstrFindWhat;
    LPSTR lpstrReplaceWith;
    UINT wFindWhatLen;
    UINT wReplaceWithLen;
    LPARAM lCustData;
    UINT (CALLBACK* lpfnHook)(HWND, UINT, WPARAM,
                              LPARAM);
    LPCSTR    lpTemplateName;
} FINDREPLACE;
```

Rückgabewert Der Rückgabewert ist das Fenster-Handle des Dialogfensters, wenn die Funktion erfolgreich ist. Er ist *NULL*, falls ein Fehler aufgetreten ist. Anwendungen können dieses Fenster-Handle benutzen, um mit dem Dialogfenster zu kommunizieren oder um es zu schließen.

Fehler Verwenden Sie die Funktion *CommDlgExtendedError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

CDERR_FINDRESFAILURE
CDERR_INITIALIZATION
CDERR_LOADRESFAILURE
CDERR_LOADSTRFAILURE
CDERR_LOCKRESFAILURE
CDERR_MEMALLOCFAILURE
CDERR_MEMLOCKFAILURE
CDERR_NOHINSTANCE
CDERR_NOHOOK
CDERR_NOTEMPLATE

CDERR_STRUCTSIZE
FRERR_BUFFERLENGTHZERO

Hinweis

Die Dialogfensterfunktion für das Dialogfenster *Suchen* übergibt Benutzeranforderungen durch spezielle Botschaften an die Anwendung. Der Parameter *lParam* dieser Botschaften enthält einen Zeiger auf eine *FINDREPLACE*-Struktur. Die Prozedur sendet die Botschaften an das Fenster, das durch das Element *hwndOwner* der *FINDREPLACE*-Datenstruktur bestimmt ist. Eine Anwendung kann Bezeichner für diese Botschaften registrieren, wenn der String commdlg_FindReplace in einem Aufruf der Funktion *RegisterWindowMessage* angegeben wird.

Damit die Taste Tab korrekt arbeitet, müssen alle Anwendungen, die *FindText* aufrufen, auch die Funktion *IsDialogMessage* in ihrer Hauptbotschaftsschleife aufrufen. Die Funktion *IsDialogMessage* gibt einen Wert zurück der angibt, ob sich die jeweilige Botschaft auf das Dialogfenster *Suchen* bezieht.

Wenn die durch das Element *lpfnHook* der *FINDREPLACE*-Struktur bezeichnete Filterfunktion die *WM_CTLCOLOR*-Botschaft bearbeitet, muß diese Funktion das Handle des Pinsels zurückgeben, der zum Ausmalen des Hintergrunds des Steuerelements benutzt werden soll.

Beispiel

Das folgende Beispiel initialisiert eine *FINDREPLACE*-Struktur und ruft die Funktion *FindText* auf, um das Dialogfenster *Suchen* anzuzeigen:

```
FINDREPLACE fr;

/* Allen Feldern der Struktur wird Null zugewiesen. */

memset(&fr, 0, sizeof(FINDREPLACE));

fr.lstructsize = sizeof(FINDREPLACE);
fr.hwndOwner = hwnd;
fr.lpstrFindWhat = szFindWhat;
fr.wFindWhatLen = sizeof(szFindWhat);

hDlg = FindText(&fr);

break;
```

Zusätzlich zur Initialisierung der Elemente der *FINDREPLACE*-Struktur und zum Aufruf der Funktion *FindText* müssen Anwendungen die spezifische *FINDMSGSTRING*-Botschaft registrieren und die Botschaften des Dialogfensters bearbeiten.

Das folgende Beispiel registriert die Botschaft mit Hilfe der Funktion *RegisterWindowsMessage:*

```
UINT uFindReplaceMsg;

/* Registriert die FindReplace-Botschaft. */

uFindReplaceMsg = RegisterWindowMessage(FINDMSGSTRING);
```

Nachdem Anwendungen die *FINDMSGSTRING*-Botschaft registriert haben, können sie Botschaften unter Verwendung des Rückgabewerts von *RegisterWindowMessage* bearbeiten. Das folgende Beispiel bearbeitet Botschaften für das Dialogfenster *Suchen* und ruft anschließend eine eigene *SearchFile*-Funktion auf, um den gesuchten String zu lokalisieren:

```
LRESULT CALLBACK MainWndProc(HWND hwnd, UINT msg,
    WPARAM wparam, LPARAM lParam
{
    FINDREPLACE FAR* lpfr;

    if (msg == uFindReplaceMsg) {
        lpfr = (FINDREPLACE FAR*) lParam;
        SearchFile((BOOL) (lpfr->Flags & FR_DOWN),
            (BOOL) (lpfr->Flags & FR_MATCHCASE));
        return 0;

    }
}
```

Siehe auch *IsDialogMessage, RegisterWindowMessage, ReplaceText*

FMExtensionProc 3.1

Syntax

```
#include <wfext.h>
HMENU FAR PASCAL FMExtensionProc(hwnd, wMsg, lParam)

HWDN hwnd;     /* Handle des Fensters der Erweiterung */
WORD wMsg;     /* Bezeichner des Menüelements oder
                  der Botschaft                       */
LONG lParam;   /* zusätzliche Botschaftsinformationen */
```

Beschreibung Die Funktion *FMExtensionProc* ist eine anwendungsdefinierte Callback-Funktion. Sie bearbeitet Menübefehle und Botschaften, die an eine Erweiterungs-DLL des Datei-Managers übergeben werden.

Parameter *hwnd:* Bezeichnet das Fenster des Datei-Managers. Eine Erweiterungs-DLL sollte dieses Handle benutzen, um das übergeordnete Fenster von Dialogfenstern oder Benachrichtigungsfenstern anzugeben, die möglicherweise von der DLL angezeigt werden. Anforderungsbotschaften müssen an den Datei-Manager übergeben werden.

wMsg: Bestimmt die Botschaft. Dieser Parameter kann folgende Werte annehmen:

Wert	Bedeutung
1-99	Bezeichner des Menüelements, das der Benutzer ausgewählt hat.

Wert	Bedeutung
FMEVENT_INITMENU	Der Benutzer hat das Menü der Erweiterung ausgewählt.
FMEVENT_LOAD	Der Datei-Manager lädt die Erweiterungs-DLL .
FMEVENT_SELCHANGE	Die Auswahl im Verzeichnisfenster oder im Fenster *Suchergebnis* des Datei-Managers hat sich geändert.
FMEVENT_UNLOAD	Der Datei-Manager entfernt die Erweiterungs-DLL aus dem Speicher.
FMEVENT_USER_REFRESH	Der Benutzer hat den Befehl *Aktualisieren* aus dem Menü *Fenster* ausgewählt.

lParam: Bestimmt 32 Bits zusätzlicher, botschaftsabhängiger Informationen.

Rückgabewert Die Callback-Funktion sollte das Ergebnis der Botschaftsbearbeitung zurückgeben. Der tatsächliche Rückgabewert hängt von der Botschaft ab, die bearbeitet wird.

Hinweis Wenn der Datei-Manager die Funktion *FMExtensionProc* aufruft, wartet er mit der Aktualisierung seiner Verzeichnisfenster, bis die Funktion zurückkehrt. Dies ermöglicht es der Erweiterung, eine große Zahl von Dateioperationen auszuführen, ohne ständig durch das Neuzeichnen des Datei-Managers unterbrochen zu werden. Die Erweiterung muß keine *FM_REFRESH_WINDOWS*-Botschaft übergeben, um dem Datei-Manager mitzuteilen, daß er seine Fenster neu zeichnen muß.

FreeAllGDIMem 3.1

Syntax
```
#include <stress.h>
void FreeAllGDIMem(void)
```

Beschreibung Die Funktion *FreeAllGDIMem* gibt den gesamten Speicher frei, der von der Funktion *AllocGDIMem* reserviert wurde.

Parameter Diese Funktion hat keine Parameter.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Siehe auch *AllocGDIMem*

FreeAllMem 3.1

Syntax
```
#include <stress.h>
void FreeAllMem(void)
```

Beschreibung Die Funktion *FreeAllMem* gibt den gesamten Speicher frei, der von der Funktion *AllocMem* reserviert wurde.

Parameter Diese Funktion hat keine Parameter.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Siehe auch *AllocMem*

FreeAllUserMem 3.1

Syntax
```
#include <stress.h>
void FreeAllUserMem(void)
```

Beschreibung Die Funktion *FreeAllUserMem* gibt den gesamten Speicher frei, der von der Funktion *AllocUserMem* reserviert wurde.

Parameter Diese Funktion hat keine Parameter.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Siehe auch *AllocUserMem*

GetAspectRatioFilterEx 3.1

Syntax
```
BOOL GetAspectRatioFilterEx(hdc, lpAspectRatio)

HDC hdc;                /* Handle des Gerätekontexts */
SIZE FAR* lpAspectRatio; /* Zeiger auf SIZE-Struktur  */
```

Beschreibung Die Funktion *GetAspectRatioFilterEx* ruft die Einstellung des aktuellen Seitenverhältnisfilters ab. Das Seitenverhältnis ist das Verhältnis von Breite und Höhe der Pixel eines Geräts. Informationen über das Seitenverhältnis eines Geräts werden bei der Erzeugung, Auswahl und Darstellung von Schriften benötigt. Windows stellt einen speziellen Filter, den Seitenverhältnisfilter, zur Verfügung. Mit diesem können Schriften, die für ein spezielles Seitenverhältnis entworfen wurden, aus den verfügbaren Schriften ausgewählt werden. Der Filter benutzt das Seitenverhältnis, das in der Funktion *SetMapperFlags* angegeben wurde.

Parameter *hdc:* Bezeichnet den Gerätekontext, der das angegebene Seitenverhältnis enthält.

lpAspectRatio: Zeiger auf eine *SIZE*-Struktur, in der der aktuelle Seitenverhältnisfilter zurückgegeben wird.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Siehe auch *SetMapperFlags*

GetBitmapDimensionEx 3.1

Syntax

```
BOOL GetBitmapDimensionEx(hBitmap, lpDimension)

HBITMAP hBitmap;        /* Handle des Bitmap          */
SIZE FAR* lpDimension;  /* Adresse der SIZE-Struktur  */
```

Beschreibung Die Funktion *GetBitmapDimensionEx* gibt die Abmessungen eines Bitmap zurück, die zuvor mit der Funktion *SetBitmapDimensionEx* gesetzt wurden. Sind keine Abmessungen gesetzt worden, wird der voreingestellte Wert 0,0 zurückgegeben.

Parameter *hBitmap:* Bezeichnet das Bitmap.

lpDimension: Zeigt auf eine *SIZE*-Struktur, in der die Abmessungen zurückgegeben werden. Die *SIZE*-Struktur hat folgende Form:

```
typedef struct tagSIZE {
    int  cx;
    int  cy;
} SIZE;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Siehe auch *SetBitmapDimensionEx*

GetBoundsRect 3.1

Syntax

```
UINT GetBoundsRect(hdc, lprcBounds, flags)

HDC hdc;              /* Handle des Gerätekontexts   */
RECT FAR* lprcBounds; /* Adresse der Struktur für
                         das umgebende Rechteck      */
UINT flags;           /* Bestimmt die zurückzu-
                         gebende Information         */
```

Beschreibung Die Funktion *GetBoundsRect* gibt das aktuelle akkumulierte umgebende Rechteck für den angegebenen Gerätekontext zurück.

Windows verwaltet zwei umgebende Rechtecke. Eins ist für die Anwendung bestimmt, das andere ist für Windows reserviert. Anwendungen können ihr eigenes Rechteck abfragen und setzen, das Windows-Rechteck dagegen nur abfragen.

Parameter *hdc:* Bezeichnet den Gerätekontext, für den das umgebende Rechteck zurückgegeben werden soll.

lprcBounds: Zeigt auf einen Puffer, der das aktuelle umgebende Rechteck aufnehmen soll. Das Rechteck der Anwendung wird in virtuellen Koordinaten, das Windows-Rechteck in Bildschirmkoordinaten zurückgegeben.

flags: Bestimmt den Typ der zurückzugebenden Informationen. Dieser Parameter kann einen oder beide der folgenden Werte annehmen:

Wert	Bedeutung
DCB_RESET	Das umgebende Rechteck wird gelöscht, sobald seine Daten zurückgegeben wurden.
DCB_WINDOWMGR	Fragt das umgebende Rechteck von Windows anstelle des Rechtecks der Anwendung ab.

Rückgabewert Der Rückgabewert bestimmt den aktuellen Status des umgebenden Rechtecks. Möglich ist eine Kombination der folgenden Werte:

Wert	Bedeutung
DCB_ACCUMULATE	Das umgebende Rechteck wird gerade akkumuliert.
DCB_RESET	Das umgebende Rechteck ist leer.
DCB_SET	Das umgebende Rechteck ist nicht leer.
DCB_ENABLE	Die Akkumulation ist eingeschaltet.
DCB_DISABLE	Die Akkumulation ist ausgeschaltet.

Hinweis Um sicherzustellen, daß das umgebende Rechteck leer ist, müssen die beiden Bits *DCB_RESET* und *DCB_ACCUMULATE* im Rückgabewert überprüft werden. Wenn *DCB_RESET* gesetzt und *DCB_ACCUMULATE* nicht gesetzt ist, ist das umgebende Rechteck leer.

Siehe auch *SetBoundsRect*

GetBrushOrgEx 3.1

Syntax

```
BOOL GetBrushOrgEx(hDC, lpPoint)

HDC hDC;             /* Handle des Gerätekontexts     */
POINT FAR* lpPoint;  /* Adresse der Struktur für den
                        Ursprung des Pinsels          */
```

Beschreibung Die Funktion *GetBrushOrgEx* ruft den aktuellen Ursprung des Pinsels im angegebenen Gerätekontext ab.

Parameter *hDC:* Bezeichnet den Gerätekontext.

lpPoint: Zeigt auf eine *POINT*-Struktur, in die die Gerätekoordinaten des Pinselursprungs kopiert werden. Die *POINT*-Struktur hat folgende Form:

```
typedef struct tagPOINT {   /* pt */
    int x;
    int y;
} POINT;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Der anfängliche Pinselursprung befindet sich an den Koordinaten (0,0).

Siehe auch *SetBrushOrg*

GetCharABCWidths 3.1

Syntax

```
BOOL GetCharABCWidths(hdc, uFirstChar, uLastChar, lpabc)

HDC hdc;          /* Handle des Gerätekontexts        */
UINT uFirstChar   /* erstes Zeichen des abzufragenden
                     Bereichs                         */
UINT uLastChar    /* letztes Zeichen des abzufragenden
                     Bereichs                         */
LPABC lpabc       /* Zeiger auf die ABC-Strukturen für
                     die Breiten                      */
```

Beschreibung Die Funktion *GetCharABCWidths* ruft die Breiten (Dicken) aufeinanderfolgender Zeichen in einem bestimmten Bereich der aktuellen TrueType-Schrift ab. Die Breiten werden in virtuellen Einheiten zurückgegeben.

Parameter *hdc:* Bezeichnet den Gerätekontext.

uFirstChar: Bestimmt das erste Zeichen eines Bereichs von Zeichen der aktuellen Schrift, deren Zeichenbreiten zurückgegeben werden.

uLastChar: Bestimmt das letzte Zeichen eines Bereichs von Zeichen der aktuellen Schrift, deren Zeichenbreiten zurückgegeben werden.

lpabc: Zeigt auf ein Array aus *ABC*-Strukturen, die die von der Funktion zurückgegebenen Zeichenbreiten aufnehmen. Dieses Array muß mindestens soviel *ABC*-Strukturen enthalten, wie sich Zeichen in dem durch die Parameter *uFirstChar* und *uLastChar* angegebenen Bereich befinden.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich war. Andernfalls ist der Rückgabewert Null.

Hinweis Die TrueType-Rasterung setzt die Zeichen gemäß der *ABC*-Strukturen, sobald eine bestimmte Punktgröße ausgewählt wurde. »A« ist der Zwischenraum, der zur aktuellen Position hinzugefügt wird, bevor der Zeichenkörper positioniert wird.

»B« ist die Breite des Zeichenkörpers. »C« ist der Zwischenraum, der zur aktuellen Position hinzugefügt wird, nachdem der Zeichenkörper gesetzt wurde. Der gesamte Vorschub ergibt sich aus A + B + C.

Ermittelt die Funktion *GetCharABCWidths* negative Werte für »A« oder »C« für ein bestimmtes Zeichen, so handelt es sich dabei um ein Zeichen, das rechts oder links Unterschneidungen aufweist.

Um die ABC-Breiten in Maßeinheiten zu konvertieren, die der Schriftdefinition zugrunde liegen, sollten Anwendungen eine Schrift erstellen, deren Schriftgröße - angegeben im Element *lfHeight* der *LOGFONT*-Struktur - gleich dem Wert ist, der im Element *ntmSizeEM* der *NEWTEXTMETRIC*-Struktur gespeichert ist. Der Wert des Elements *ntmSizeEM* kann mit Hilfe der Funktion *EnumFontFamilies* abgerufen werden.

Die ABC-Breiten des Standardzeichens werden für alle Zeichen benutzt, die nicht im angegebenen Bereich der aktuell gewählten Schrift liegen.

Um die Breiten von Zeichen in Nicht-TrueType-Schriften abzurufen, sollten Anwendungen die Funktion *GetCharWidths* benutzen.

Siehe auch *EnumFontFamilies, GetCharWidth*

GetClipCursor 3.1

Syntax

```
void GetClipCursor(lprc)

RECT FAR* lprc;     /* Adresse der Struktur für
                       die Rechteckkoordinaten      */
```

Beschreibung Die Funktion *GetClipCursor* ruft die Bildschirmkoordinaten des Rechtecks ab, auf das der Mauszeiger durch einen vorausgehenden Aufruf der Funktion *ClipCursor* beschränkt wurde.

Parameter *lprc:* Zeigt auf eine *RECT*-Datenstruktur, die die Koordinaten des eingrenzenden Rechtecks aufnimmt. Ist der Mauszeiger nicht auf ein bestimmtes Rechteck beschränkt, erhält die Struktur die Abmessungen des Bildschirms. Die *RECT*-Struktur hat folgende Form:

```
typedef struct tagRECT {   /* rc */
   int left;
   int top;
   int right;
   int bottom;
} RECT;
```

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Siehe auch *ClipCursor, GetCursorPos*

GetCurrentPositionEx 3.1

Syntax

```
BOOL GetCurrentPositionEx(hdc, lpPoint)

HDC hdc;                /* Handle des Gerätekontexts */
POINT FAR* lpPoint;     /* Zeiger auf POINT-Struktur */
```

Beschreibung Die Funktion *GetCurrentPositionEx* ruft die aktuelle Position in virtuellen Koordinaten ab.

Parameter *hdc:* Bezeichnet den Gerätekontext, aus dem die aktuelle Position abgerufen werden soll.

lpPoint: Zeigt auf eine *POINT*-Struktur, die mit der aktuellen Position gefüllt wird.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

GetCursor 3.1

Syntax

```
HCURSOR GetCursor(void)
```

Beschreibung Die Funktion *GetCursor* ruft das Handle des aktuellen Mauszeigers ab.

Parameter Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert ist das Handle des aktuellen Mauszeigers oder *NULL*, wenn kein Mauszeiger existiert.

Siehe auch *SetCursor*

GetDCEx 3.1

Syntax

```
HDC GetDCEx(hwnd, hrgnClip, fdwOptions)

register HWND hwnd;   /* Fenster, in dem gezeichnet
                         wird                        */
HRGN hrgnClip;        /* Clipping-Region, kann
                         kombiniert werden           */
DWORD fdwOptions;     /* Gerätekontextoptionen       */
```

Beschreibung Die Funktion *GetDCEx* ruft das Handle eines Gerätekontexts für das angegebene Fenster ab. Der Gerätekontext kann in nachfolgenden Aufrufen von Funktionen der grafischen Geräteschnittstelle (GDI) benutzt werden, um in der Client-Region zu zeichnen.

Diese Funktion, eine Erweiterung der Funktion *GetDC*, gibt einer Anwendung eine erweiterte Kontrolle über das Clipping eines Gerätekontexts eines Fensters.

Parameter *hwnd:* Bezeichnet das Fenster, in dem gezeichnet wird.

hrgnClip: Bezeichnet eine Clipping-Region, die mit dem sichtbaren Bereich des Client-Fensters kombiniert werden kann.

fdwOptions: Bestimmt, wie der Gerätekontext erzeugt wird. Der Parameter kann eine Kombination der folgenden Werte sein:

Wert	Bedeutung
DCX_CACHE	Gibt einen Gerätekontext aus dem Cache anstelle des Fensters *OWNDC* oder *CLASSDC* zurück. Überschreibt *CS_OWNDC* und *CS_CLASSDC*.
DCX_CLIP-CHILDREN	Schließt die sichtbaren Regionen aller untergeordneten Fenster unterhalb des durch den Parameter *hwnd* bezeichneten Fensters aus.
DCX_CLIP-SIBLINGS	Schließt die sichtbaren Regionen aller nebengeordneten Fenster oberhalb des durch den Parameter *hwnd* bezeichneten Fensters aus.
DCX_EXCLUDE-RGN	Schließt die durch den Parameter *hrgnClip* bezeichnete Clipping-Region aus der sichtbaren Region des zurückgegebenen Gerätekontexts aus.
DCX_INTERSECT RGN	Teilt die durch den Parameter *hrgnClip* bezeichnete Clipping-Region mit der sichtbaren Region des zurückgegebenen Gerätekontexts.
DCX_LOCK-WINDOWUPDATE	Ermöglicht das Zeichnen auch dann, wenn die Funktion *LockWindowUpdate* für dieses Fenster aufgerufen wurde. Dieser Wert wird für das Zeichnen während einer Verschiebe-Operation verwendet.
DCX_PARENTCLIP	Benutzt die sichtbare Region des übergeordneten Fensters. Die Bits *WS_CLIP-CHILDREN* und *WS_PARENTDC* des übergeordneten Fensters werden ignoriert. Dieser Wert setzt den Ursprung des Geräteskontexts auf die obere linke Ecke des durch den Parameter *hwnd* bezeichneten Fensters.
DCX_WINDOW	Gibt einen Gerätekontext zurück, der dem Fensterrechteck und nicht dem Client-Rechteck entspricht.

Rückgabewert

Wenn die Funktion erfolgreich ist, wird das Handle des Gerätekontexts für das angegebene Fenster zurückgegeben. Andernfalls ist der Rückgabewert *NULL*.

Hinweis

Gehört der Gerätekontext nicht zu einer Fensterklasse, muß die Funktion *ReleaseDC* aufgerufen werden, um den Kontext nach dem Zeichnen freizugeben. Da zu einem gegebenen Zeitpunkt nur fünf allgemeine Gerätekontexte verfügbar sind, kann das Unterlassen der Freigabe eines Gerätekontexts andere Anwendungen behindern, da diese nicht auf einen Gerätekontext zugreifen können.

Ein Gerätekontext, der zur Klasse eines Fensters gehört, wird von der Funktion *GetDCEx* zurückgegeben, wenn der Klassenstil *CS_CLASSDC*, *CS_OWNDC* oder *CS_PARENTDC* in der *WNDCLASS*-Struktur angegeben war, als die Klasse registriert wurde.

Um einen zwischengespeicherten Gerätekontext zu erhalten, müssen Anwendungen *DCX_CACHE* angeben. Wenn *DCX_CACHE* nicht angegeben wird und das Fenster weder den Stil *CS_OWNDC* noch *CS_CLASSDC* hat, gibt diese Funktion *NULL* zurück.

Siehe auch

BeginPaint, *GetDC*, *GetWindowDC*, *ReleaseDC*

GetDriverInfo 3.1

Syntax

```
BOOL GetDriverInfo(hdrvr, lpdis)

HDRVR hdrvr;                /* Handle eines instal-
                               lierbaren Treibers    */
DRIVERINFOSTRUCT FAR* lpdis; /* Adresse der Struktur
                               für die Informationen */
```

Beschreibung

Die Funktion *GetDriverInfo* ruft Informationen über einen installierbaren Treiber ab.

Parameter

hdrvr: Bezeichnet den installierbaren Treiber. Dieses Handle muß mit der Funktion *OpenDriver* abgerufen werden.

lpdis: Zeigt auf eine *DRIVERINFOSTRUCT*-Struktur, die die Treiberinformationen aufnimmt. Die *DRIVERINFOSTRUCT*-Struktur hat folgende Form:

```
typedef struct tagDRIVERINFOSTRUCT {   /* drvinfst */
    UINT length;
    HDRVR hDriver;
    HINSTANCE hModule;
    char szAliasName[128];
} DRIVERINFOSTRUCT;
```

Rückgabewert

Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

GetDriverModuleHandle 3.1

Syntax

```
HINSTANCE GetDriverModuleHandle(hdrvr)

HDRVR hdrvr;   /* Handle des installierbaren Treibers */
```

Beschreibung Die Funktion *GetDriverModuleHandle* ruft das Instanzen-Handle eines Moduls ab, das einen installierbaren Treiber enthält.

Paramter *hdrvr:* Bezeichnet den installierbaren Treiber. Dieser Wert muß mit der Funktion *OpenDriver* abgerufen werden.

Rückgabewert Wenn die Funktion erfolgreich ist, gibt sie das Instanzen-Handle des Treibermoduls zurück. Andernfalls ist der Rückgabewert *NULL*.

Siehe auch *OpenDriver*

GetExpandedName 3.1

Syntax

```
#include <lzexpand.h>
int GetExpandedName(lpszSource, lpszBuffer)

LPCSTR lpszSource;  /* Bestimmt den Namen der
                       komprimierten Datei          */
LPSTR lpszBuffer;   /* Zeigt auf den Puffer, der den
                       ursrpünglichen Dateinamen
                       aufnimmt                     */
```

Beschreibung Die Funktion *GetExpandedName* ruft den ursprünglichen Namen einer komprimierten Datei ab, wenn diese Datei mit dem Hilfsprogramm COMPRESS.EXE und der Option */r* komprimiert wurde.

Parameter *lpszSource:* Zeigt auf einen String, der den Namen einer komprimierten Datei angibt.

lpszBuffer: Zeigt auf einen Puffer, der den ursprünglichen Namen der komprimierten Datei aufnimmt.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert TRUE. Andernfalls handelt es sich um einen Fehlerwert, der kleiner als Null ist. Ein möglicher Wert ist *LZERROR_BADINHANDLE*. Dieser Wert zeigt an, daß das Handle der Quelldatei ungültig war.

Beispiel Das folgende Beispiel benutzt die Funktion *GetExpandedName*, um den ursprünglichen Dateinamen einer komprimierten Datei abzurufen:

```
char szSrc[] = {"readme.cmp"}
char szFileName[128];
OFSTRUCT ofStrSrc;
```

```
OFSTRUCT ofStrDest;
HFILE hfSrcFile, hfDstFile, hfCompFile;
int cbRead;
BYTE abBuf[512];

/* Die komprimierte Quelldatei öffnen. */

hfSrcFile = OpenFile(szSrc, &ofStrSrc, OF_READ);

/*
 * Interne Datenstrukturen für die Entkomprimierung
 * initialisieren.
 */

hfCompFile = LZInit(hfSrcFile);

/* Den ursprünglichen Namen der komprimierten Datei
   abrufen. */

GetExpandedName(szSrc, szFileName);

/* Die Zieldatei mit dem Originalnamen erzeugen. */

hfDstFile = LZOpenFile(szFileName, &ofStrDest,
 OF_CREATE);

/* Die komprimierte Quelldatei in die Zieldatei
   kopieren. */

do {
    if ((cbRead = LZRead(hfCompFile, abBuf,
         sizeof(abBuf))) > 0)
        _lwrite(hfDstFile, abBuf, cbRead);
    else {
        .
        . /* Fehlerbehandlung */
        .
    }
} while (cbRead == sizeof(abBuf));

/* Dateien schließen. */

LZClose(hfSrcFile);
LZClose(hfDstFile);
```

Hinweis

Diese Funktion ruft den ursprünglichen Dateinamen aus dem Header der komprimierten Datei ab. Wenn die Quelldatei nicht komprimiert ist, wird der durch *lpszSource* bezeichnete Dateiname in den Puffer kopiert, auf den *lpszBuffer* zeigt.

Wenn die Option */r* beim Komprimieren der Datei nicht angegeben wurde, ist der String in dem durch *lpszBuffer* bezeichneten Puffer ungültig.

GetFileResource 3.1

Syntax

```
#include <ver.h>
BOOL GetFileResource (lpszFileName, lpszResType,
   lpszResID, dwFileOffset, dwResLen, lpvData)

LPCSTR lpszFileName;   /* Adresse des Puffers für den
                          Dateinamen                */
LPCSTR lpszResType;    /* Adresse des Puffers für den
                          Ressourcentyp             */
LPCSTR lpszResID;      /* Adresse des Puffers für die
                          Ressourcen-ID             */
DWORD dwFileOffset;    /* Offset der Ressource in der
                          Datei                     */
DWORD dwResLen;        /* Größe des Ressourcenpuffers */
void FAR* lpvData;     /* Adresse des Puffers für die
                          Kopie der Ressource       */
```

Beschreibung

Die Funktion *GetFileResource* kopiert die angegebene Ressource aus einer Datei in den angegebenen Puffer. Um die benötigte Puffergröße zu ermitteln, kann die Anwendung die Funktion *GetFileResourceSize* vor *GetFileResource* aufrufen.

Parameter

lpszFileName: Zeigt auf den Puffer, der den Namen der Datei mit der Ressource enthält.

lpstrResType: Zeigt auf den Wert, der durch Anwendung des Makros *MAKEINTRESOURCE* auf den numerierten Ressourcentyp erzeugt wird. Dieser Wert ist normalerweise *VS_FILE_INFO*.

lpszResID: Zeigt auf den Wert, der durch Anwendung des Makros *MAKEINTRESOURCE* auf den numerierten Ressourcenbezeichner erzeugt wird. Dieser Wert ist normalerweise *VS_VERSION_INFO*.

dwFileOffset: Bestimmt den Offset der Ressource in der Datei. Die Funktion *GetFileResourceSize* gibt diesen Wert zurück. Hat dieser Parameter den Wert Null, durchsucht die Funktion *GetFileResource* die Datei nach der Ressource.

dwResLen: Bestimmt die Größe des Puffers in Byte, den der Parameter *lpvData* bezeichnet. Die Funktion *GetFileResourceSize* gibt die benötigte Puffergröße für die Ressource zurück. Falls der Puffer nicht groß genug ist, werden die Ressourcendaten bis auf die Größe des Puffers abgeschnitten.

lpvData: Zeigt auf den Puffer, der die Kopie der Ressource aufnimmt. Ist der Puffer nicht ausreichend groß, werden die Ressourcendaten abgeschnitten.

Rückgabewert

Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich ist. Er ist Null, wenn die Datei oder die Ressource nicht

gefunden werden konnte, oder wenn ein DOS-Fehler aufgetreten ist. Die Funktion *GetFileResource* liefert keine Informationen über die Art des aufgetretenen Fehlers.

Hinweis

Wenn der Parameter *dwFileOffset* Null ist, ermittelt die Funktion *GetFileResource* die Position der Ressource mit Hilfe der Parameter *lpszResType* und *lpszResID*.

Ist *dwFileOffset* ungleich Null, behandelt *GetFileResource* diesen Wert wie den Rückgabewert von *GetFileResourceSize* und ignoriert deshalb *lpszResType* und *lpszResID*.

Siehe auch

GetFileResourceSize

GetFileResourceSize 3.1

Syntax

```
#include <ver.h>
DWORD GetFileResourceSize (lpszFileName, lpszResType,
   lpszResID, lpdwFileOffset)

LPCSTR lpszFileName;       /* Adresse des Puffers für
                              den Dateinamen          */
LPCSTR lpszResType;        /* Adresse des Puffers für
                              den Ressourcentyp       */
LPCSTR lpszResID;          /* Adresse des Puffers für
                              den Ressourcen-
                              bezeichner              */
DWORD FAR* lpdwFileOffset; /* Adresse des Ressourcen-
                              offsets in der Datei    */
```

Beschreibung

Die Funktion *GetFileResourceSize* durchsucht die angegebene Datei nach der Ressource des angegebenen Typs und mit dem angegebenen Bezeichner.

Parameter

lpszFileName: Zeigt auf den Puffer, der den Namen der Datei enthält, in der nach der Ressource gesucht werden soll.

lpszResType: Zeigt auf den Wert, der durch Anwendung des Makros *MAKEINTRESOURCE* auf den numerierten Ressourcentyp erzeugt wird. Dieser Wert ist normalerweise *VS_FILE_INFO*.

lpszResID: Zeigt auf den Wert, der durch Anwendung des Makros *MAKEINTRESOURCE* auf den numerierten Ressourcenbezeichner erzeugt wird. Dieser Wert ist normalerweise *VS_VERSION_INFO*.

lpdwFileOffset: Zeigt auf den 16-Bit-Wert, den die Funktion *GetFileResourceSize* mit dem Offset der Ressource innerhalb der Datei initialisiert.

Rückgabewert

Der Rückgabewert ist die Größe der Ressource in Byte. Er ist *NULL*, wenn die Funktion die Datei nicht finden konnte, mit der Datei keine Ressourcen verknüpft sind, oder wenn ein

DOS-Fehler aufgetreten ist. Die Funktion *GetFileResourceSize* liefert keine Informationen über die Art des aufgetretenen Fehlers.

Siehe auch *GetFileResource*

GetFileTitle 3.1

Syntax

```
#include <commdlg.h>
int GetFileTitle(lpszFile, lpszTitle, cbBuf)

LPCSTR lpszFile;       /* Zeiger auf den Dateinamen
                          (inklusive Laufwerk und
                          Verzeichnis)              */
LPSTR lpszTitle;       /* Adresse des Puffers für den
                          Dateinamen                */
UINT cbBuf;            /* Länge des Puffers         */
```

Beschreibung Die Funktion *GetFileTitle* liefert im Parameter *lpszTitle* den Namen der über den Parameter *lpszFile* bezeichneten Datei zurück.

Parameter *lpszFile:* Zeigt auf den Namen und die Position einer DOS-Datei.

lpszTitle: Zeigt auf den Puffer, in den die Funktion den Dateinamen kopiert.

wBufSize: Gibt die Länge des Puffers an, auf den *lpszTitle* zeigt.

Rückgabewert Der Rückgabewert ist Null, wenn die Funktion erfolgreich ausgeführt werden konnte. Ist der Dateiname ungültig, liefert die Funktion einen negativen Wert zurück. Wenn der durch *lpszTitle* bezeichnete Puffer zu klein ist, ist der Rückgabewert eine positive Zahl, die die benötigte Puffergröße angibt.

Hinweis Die Funktion gibt einen Fehlerwert zurück, wenn der durch den Parameter *lpszFile* bezeichnet Puffer folgendes enthält:

- einen leeren String;
- einen String, der ein Platzhalterzeichen (*) oder eine öffnende ([) bzw. schließende (]) eckige Klammer enthält;
- einen String, der mit einem Doppelpunkt (:), einem Schrägstrich (/) oder einem Backslash (\) endet;
- einen String, dessen Länge die Größe des Puffers überschreitet;
- ein ungültiges Zeichen (Leerzeichen oder nicht druckbares Zeichen).

Die für den Puffer benötigte Größe muß das abschließende Nullzeichen berücksichtigen.

GetFileVersionInfo 3.1

Syntax

```
#include <ver.h>
BOOL GetFileVersionInfo(lpszFileName, handle, cbBuf,
   lpvData)

LPCSTR lpszFileName;   /* Adresse des Puffers für den
                          Dateinamen                */
DWORD handle;          /* Informationen über die
                          Dateiversion              */
DWORD cbBuf;           /* Größe des Puffers         */
void FAR* lpvData;     /* Adresse des Puffers für
                          die Versionsdaten         */
```

Beschreibung

Diese Funktion gibt Versionsinformationen über die angegebene Datei zurück. Anwendungen müssen die Funktion *GetFileVersionInfoSize* vor *GetFileVersionInfo* aufrufen, um das entsprechende Handle zu erhalten, wenn dieses ungleich *NULL* ist.

Parameter

lpszFileName: Zeigt auf den Puffer, der den Namen der Datei enthält.

handle: Bezeichnet die Informationen über die Dateiversion. Die Funktion *GetFileVersionInfoSize* gibt dieses Handle zurück. Ist der Parameter *handle NULL*, durchsucht die Funktion *GetFileVersionInfo* die Datei nach den Versionsinformationen.

cbBuf: Bestimmt die Größe des Puffers in Byte, der durch *lpvData* bezeichnet wird. Die Funktion *GetFileVersionInfoSize* gibt die Puffergröße zurück, die für die Informationen über die Dateiversion benötigt wird. Falls der Puffer zu klein ist, werden die Informationen über die Dateiversion auf die Puffergröße gekürzt.

lpvData: Zeigt auf den Puffer, der die Informationen über die Dateiversion aufnimmt. Dieser Parameter wird in nachfolgenden Aufrufen der Funktion *VerQueryValue* benötigt.

Rückgabewert

Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich ausgeführt wurde. Der Rückgabewert Null zeigt an, daß die Datei nicht existiert oder der Parameter *handle* ungültig ist. Die Funktion *GetFileVersionInfo* gibt keine Informationen über die Art des aufgetretenen Fehlers zurück.

Hinweis

Die Informationen über die Dateiversion sind in einem *VS_VERSION_INFO*-Block angeordnet.

In der aktuellen Version erkennt die Funktion *GetFileVersionInfo* nur die Versionsinformationen, die vom Microsoft Resourcencompiler (RC) erzeugt wurden.

Siehe auch

GetFileVersionInfoSize, VerQueryValue

GetFileVersionInfoSize 3.1

Syntax

```
#include <ver.h>
DWORD GetFileVersionInfoSize(lpszFileName, lpdwHandle)

LPCSTR lpszFileName;      /* Adresse des Puffers für
                             den Dateinamen          */
DWORD FAR* lpdwHandle;    /* Adresse des Handle
                             der Informationen       */
```

Beschreibung Die Funktion *GetFileVersionInfoSize* ermittelt, ob Versionsinformationen aus der angegebenen Datei abgerufen werden können. Sind die Versionsinformationen verfügbar, gibt die Funktion *GetFileVersionInfoSize* die benötigte Größe des Puffers zurück, der die Versionsinformationen aufnehmen soll. Außerdem stellt die Funktion ein Handle zur Verfügung, das in späteren Aufrufen der Funktion *GetFileVersionInfo* benutzt werden kann.

Parameter *lpszFileName:* Zeigt auf den Puffer, der den Dateinamen enthält.

lpdwHandle: Zeigt auf einen 32-Bit-Wert, den die Funktion *GetFileVersionInfoSize* mit dem Handle auf die Informationen zur Dateiversion initialisiert. Die Funktion *GetFileVersionInfo* kann dieses Handle benutzen.

Rückgabewert Der Rückgabewert ist die Größe des Puffers in Byte, die zum Speichern der Versionsinformationen benötigt wird. Der Rückgabewert ist *NULL*, wenn die Funktion die Datei oder die Versionsinformationen nicht findet oder ein DOS-Fehler auftritt. Die Funktion *GetFileVersionInfoSize* liefert keine Informationen über die Art des aufgetretenen Fehlers.

Hinweis Die Informationen über die Dateiversion sind in einem *VS_VERSION_INFO*-Block angeordnet.

Siehe auch *GetFileVersionInfo*

GetFontData 3.1

Syntax

```
DWORD GetFontData(hdc, dwTable, dwOffset, lpvBuffer,
   cbData)

HDC hdc;                 /* Handle des Gerätekontexts  */
DWORD dwTable;           /* abzufragende metrische
                            Tabelle                    */
DWORD dwOffset;          /* Offset in die abgefragte
                            Tabelle                    */
void FAR* lpvBuffer;     /* Adresse des Puffers für die
                            Schriftdaten               */
```

```
DWORD cbData;          /* Länge der abzufragenden
                          Daten                    */
```

Beschreibung Die Funktion *GetFontData* ruft die metrischen Daten einer skalierbaren Schrift aus ihrer Schriftdatei ab. Die abzufragenden Daten werden bestimmt, indem ein Offset in die Schriftdatei und die Länge der abzufragenden Daten angegeben wird.

Parameter *hdc:* Bezeichnet den Gerätekontext.

dwTable: Bestimmt den Namen der metrischen Tabelle, die zurückgegeben werden soll. Dieser Parameter kann jede der von Microsoft dokumentierten metrischen Tabellen für TrueType-Schriftdateien sein. Hat dieser Parameter den Wert Null, werden die Informationen beginnend am Anfang der Datei abgerufen.

dwOffset: Bestimmt den Offset in die Tabelle, ab dem die Daten abgerufen werden sollen. Ist der Wert Null, wird mit dem Abrufen der Informationen am Anfang der Tabelle begonnen, die vom Parameter *dwTable* bezeichnet wird. Ist dieser Wert größer oder gleich der Größe der Tabelle, gibt *GetFontData* Null zurück.

lpvBuffer: Zeigt auf den Puffer, der die metrischen Daten der Schrift aufnimmt. Wenn dieser Wert *NULL* ist, gibt die Funktion die Größe des Puffers zurück, die für die Schriftdaten in der durch *dwTable* bezeichneten Tabelle benötigt wird.

cbData: Bestimmt die Länge der abzurufenden Daten in Byte. Wenn dieser Parameter Null ist, gibt *GetFontData* den Umfang der Daten in der durch *dwTable* bezeichneten Tabelle zurück.

Rückgabewert Der Rückgabewert bestimmt die Anzahl der Byte, die in dem Puffer zurückgegeben werden, auf den *lpvBuffer* zeigt. Er ist -1, wenn die Funktion fehlschlägt.

Hinweis Anwendungen können die Funktion *GetFontData* benutzen, um eine TrueType-Schrift zusammen mit einem Dokument zu speichern. Dazu stellt die Anwendung fest, ob die Schrift eingebettet ist und ruft dann die komplette Schriftdatei ab. In diesem Fall wird für die Parameter *dwTable*, *dwOffset* und *cbData* Null angegeben.

Anwendungen ermitteln mit Hilfe des Elements *otmfsType* der *OUTLINETEXTMETRIC*-Struktur, ob eine Schrift eingebettet ist. Ist das Bit 1 in *otmfsType* gesetzt, kann diese Schrift nicht eingebettet werden. Ist Bit 1 nicht gesetzt, ist die Einbettung möglich. Wenn zusätzlich das Bit 2 gesetzt ist, handelt es sich um eine Nur-Lesen-Einbettung.

Wendet eine Anwendung diese Funktion auf eine nichtskalierbare Schrift an, gibt *GetFontData* -1 zurück.

Beispiel

Im folgenden Beispiel wird eine komplette TrueType-Schriftdatei abgerufen:

```
HGLOBAL hglb;
DWORD dwSize;
void FAR* lpvBuffer;

/* Dateigröße ermitteln */
dwSize = GetFontData(hdc, NULL, 0L, NULL, 0L);

/* Speicher reservieren */
hglb = GlobalAlloc(GPTR, dwSize);
lpvBuffer = GlobalLock(hglb);
/* Daten abrufen */
GetFontData(hdc, NULL, 0L, lpvBuffer, dwSize);
```

Mit dem folgenden Beispiel wird eine komplette TrueType-Schriftdatei in 4 KByte-Schritten abgerufen:

```
#define SIZE 4096
BYTE Buffer[SIZE];
DWORD dwOffset;
DWORD dwSize;
dwOffset = 0L;
while(dwSize = GetFontData(hdc, NULL, dwOffset,
                           Buffer, SIZE)) {
  .
  . /* Daten im Puffer bearbeiten */
  .
  dwOffset += dwSize;
}
```

Das folgende Beispiel ruft eine Tabelle einer TrueType-Schrift ab:

```
HGLOBAL hglb;
DWORD dwSize;
void FAR* lpvBuffer;

LPSTR lpszTable;
DWORD dwTable;

lpszTable = "cmap";
/* DWORD-Typ erzeugen */
dwTable   = *(LPDWORD) lpszTable;

/* Tabellengröße ermitteln */
dwSize = GetFontData(hdc, dwTable, 0L, NULL, 0L);

/* Speicher reservieren */
hglb = GlobalAlloc(GPTR, dwSize);
lpvBuffer = GlobalLock(hglb);
/* Daten abrufen */
GetFontData(hdc dwTable, 0L, lpvBuffer, dwSize);
```

Siehe auch *GetOutlineTextMetrics*

GetFreeFileHandles 3.1

Syntax
```
#include <stress.h>
int GetFreeFileHandles(void)
```

Beschreibung Die Funktion *GetFreeFileHandles* gibt die Anzahl der Datei-Handles zurück, die für die aktuelle Instanz verfügbar sind.

Parameter Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert ist die Anzahl der Datei-Handles, die für die aktuelle Instanz verfügbar sind.

GetFreeSystemResources 3.1

Syntax
```
UINT GetFreeSystemResources(fuSysResource)

UINT fuSysResource;   /* Typ der zu überprüfenden
                         Ressource                  */
```

Beschreibung Die Funktion *GetFreeSystemResources* gibt den freien Platz für Systemressourcen in Prozent zurück.

Parameter *fuSysResource:* Bestimmt den Typ der zu überprüfenden Ressource. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
GFSR_SYSTEM-RESOURCES	Gibt den freien Platz für Systemressourcen in Prozent zurück.
GFSR_GDI-RESOURCES	Gibt den freien Platz für GDI-Ressourcen in Prozent zurück. GDI-Ressourcen schließen Handles für Gerätekontexte, Pinsel, Stifte, Regionen, Schriften und Bitmaps ein.
GFSR_USER-RESOURCES	Gibt den freien Platz für USER-Ressourcen in Prozent zurück. Diese Ressourcen schließen Fenster- und Menü-Handles ein.

Rückgabewert Der Rückgabewert bestimmt den freien Platz für Ressourcen in Prozent, wenn die Funktion erfolgreich ist.

Hinweis Da der Rückgabewert dieser Funktion nicht sicherstellt, daß eine Anwendung ein neues Objekt erzeugen kann, sollten Anwendungen diese Funktion nicht benutzen, um zu ermitteln, ob sie das Objekt erzeugen können.

Siehe auch *GetFreeSpace*

GetGlyphOutline 3.1

Syntax

```
DWORD GetGlyphOutline(hdc, uChar, fuFormat, lpgm,
   cbBuffer, lpBuffer, lpmat2)

HDC hdc;                /* Handle des Gerätekontexts   */
UINT uChar;             /* abzufragendes Zeichen       */
UINT fuFormat;          /* Format der zurückzugebenden
                           Daten                       */
LPGLYPHMETRICS lpgm;    /* Adresse der Struktur mit den
                           Abmessungen des Zeichen-
                           körpers                     */
DWORD cbBuffer;         /* Größe des Datenpuffers      */
void FAR* lpBuffer;     /* Adresse des Puffers für die
                           Umrißdaten                  */
LPMAT2 lpmat2;          /* Adresse der Struktur mit
                           der Transformationsmatrix   */
```

Beschreibung Die Funktion *GetGlyphOutline* ruft die Umrißkurve oder das Bitmap eines Outline-Zeichens der aktuellen Schrift ab.

Parameter *hdc:* Bezeichnet den Gerätekontext.

uChar: Bestimmt das Zeichen, für das die Informationen angefordert werden.

fuFormat: Bestimmt das Format, in dem die Funktion die Informationen zurückgibt. Möglich sind folgende Werte:

Wert	Bedeutung
GGO_BITMAP	Das Bitmap des Zeichenkörpers wird zurückgegeben. Sobald die Funktion zurückkehrt, enthält der durch den Parameter *lpBuffer* bezeichnete Puffer ein Bitmap (ein Bit pro Pixel), dessen Zeilen an *DWORD*-Adressen ausgerichtet sind.
GGO_NATIVE	Die Punktdaten der Kurve werden im ursprünglichen Format der Rasterung unter Verwendung der gerätespezifischen Einheiten zurückgegeben. Ist dieser Wert angegeben, werden alle Transformationen ignoriert, die im Parameter *lpmat2* festgelegt sind.

Hat dieser Parameter den Wert Null, füllt die Funktion eine *GLYPHMETRICS*-Struktur, gibt aber nicht die Umrißdaten des Zeichenkörpers zurück.

lpgm: Zeigt auf eine *GLYPHMETRICS*-Struktur. Diese beschreibt die Ausrichtung des Zeichenkörpers in der Zeichenzelle. Die *GLYPHMETRICS*-Struktur hat folgende Form:

```
typedef struct tagGLYPHMETRICS {  /* gm */
    UINT gmBlackBoxX;
    UINT gmBlackBoxY;
    POINT gmptGlyphOrigin;
    int gmCellIncX;
    int gmCellIncY;
} GLYPHMETRICS;
```

cbBuffer: Bestimmt die Größe des Puffers, in den die Funktion die Informationen über das Outline-Zeichen kopiert. Wenn dieser Parameter den Wert Null hat und *fuFormat* auf *GGO_BITMAP* oder *GGO_NATIVE* gesetzt ist, gibt die Funktion die für diesen Puffer benötigte Größe zurück.

lpBuffer: Zeigt auf einen Puffer, in den die Funktion Informationen über das Outline-Zeichen kopiert. Wenn der Parameter *fuFormat* den Wert *GGO_NATIVE* hat, werden die Informationen in *TTPOLYGONHEADER*- und *TTPOLYCURVE*-Strukturen kopiert. Hat dieser Parameter den Wert *NULL* und der Parameter *fuFormat* den Wert *GGO_BITMAP* oder *GGO_NATIVE*, gibt die Funktion die für diesen Puffer benötigte Größe zurück.

lpmat2: Zeigt auf eine *MAT2*-Struktur, die eine Transformationsmatrix für das Zeichen enthält. *lpmat2* darf nicht *NULL* sein, auch wenn für den Parameter *fuFormat GGO_NATIVE* angegeben ist. Die *MAT2*-Struktur hat folgende Form:

```
typedef struct tagMat2 {  /* mat2 */
    FIXED eM11;
    FIXED eM12;
    FIXED eM21;
    FIXED eM22;
} MAT2;
```

Rückgabewert

Wenn der Parameter *cbBuffer* Null oder der Parameter *lpBuffer* *NULL* ist, ist der Rückgabewert die benötigte Größe für den Puffer, der die abgerufenen Informationen aufnehmen soll. Andernfalls ist der Rückgabewert positiv, wenn die Funktion erfolgreich ist und -1, wenn sie fehlschlägt.

Hinweis

Anwendungen können Zeichen rotieren, die sie im Bitmap-Format abgerufen haben. Hierfür muß eine 2-x-2-Transformationsmatrix in der Struktur angegeben werden, auf die *lpmat2* zeigt.

Der Umriß eines Zeichenkörpers wird als Reihe von Konturen zurückgegeben. Jede dieser Konturen ist durch eine *TTPOLYGONHEADER*- und die benötigte Anzahl von *TTPOLYCURVE*-Strukturen beschrieben. Sämtliche Punkte werden als *POINTFX*-Strukturen zurückgegeben. Sie repräsentieren absolute Positionen, keine relativen Verschiebungen. Der im Ele-

ment *pfxStart* der *TTPOLYGONHEADER*-Struktur angegebene Punkt ist der Startpunkt für den Umriß der jeweiligen Kontur. Die folgenden *TTPOLYCURVE*-Strukturen können Polyline- oder Spline-Records sein. Polyline-Records sind eine Reihe von Punkten, deren geradlinige Verbindung den Umriß des Zeichens ergibt. Spline-Records sind quadratische Kurven, die von TrueType verwendet werden (quadratische b-Splines).

Die Funktion *GetGlyphOutline* ruft zum Beispiel die folgenden Informationen über das Zeichen »i« in der TrueType-Schrift *Arial* ab:

```
dwrc = 88                  /* Gesamtgröße des Original-
                              puffers */

TTPOLYGONHEADER #1         /* Kontur des Punkts
                              auf i */
  cb     = 44              /* Größe der Kontur */
  dwType = 24              /* TT_POLYGON_TYPE */
  pfxStart = 1.000, 11.000

  TTPOLYCURVE #1
    wType  = TT_PRIM_LINE
    cpfx   = 3
    pfx[0] = 1.000, 12.000
    pfx[1] = 2.000, 12.000
    pfx[2] = 2.000, 11.000 /* abschließender,
                              automatischer Linienzug
                              auf pfxStart */

TTPOLYGONHEADER #2         /* Kontur des Körpers von i */
  cb     = 44
  dwType = 24              /* TT_POLYGON_TYPE */
  pfxStart = 1.000, 0.000

  TTPOLYCURVE #1
    wType  = TT_PRIM_LINE
    cpfx   = 3
    pfx[0] = 1.000, 9.000
    pfx[1] = 2.000, 9.000
    pfx[2] = 2.000, 0.000  /* abschließender,
                              automatischer Linienzug
                              auf pfxStart */
```

Siehe auch *GetOutlineTextMetrics*

GetKerningPairs 3.1

Syntax

```
int GetKerningPairs(hdc, cPairs, lpkrnpair)

HDC hdc;                    /* Handle des Geräte-
                               kontexts              */
```

```
int cPairs;                 /* Anzahl der
                               Unterschnittpaare      */
KERNINGPAIR FAR* lpkrnpair; /* Zeiger auf die
                               Strukturen für die
                               Unterschnittpaare      */
```

Beschreibung Die Funktion *GetKerningPairs* ruft die Unterschnittpaare für Zeichen der Schrift ab, die aktuell in den angegebenen Gerätekontext ausgewählt ist.

Parameter *hdc:* Bezeichnet einen Gerätekontext. Die Funktion *GetKerningPairs* ruft Unterschnittpaare für die aktuelle Schrift dieses Gerätekontexts ab.

cPairs: Bestimmt die Anzahl von *KERNINGPAIR*-Strukturen, auf die der Parameter *lpkrnpair* zeigt. Diese Funktion kopiert nicht mehr Unterschnittpaare, als in *cPairs* angegeben ist.

Die *KERNINGPAIR*-Struktur hat folgende Form:

```
typedef struct tagKERNINGPAIR {
    WORD wFirst;
    WORD wSecond;
    int iKernAmount;
} KERNINGPAIR;
```

lpkrnpair: Zeigt auf ein Array von *KERNINGPAIR*-Strukturen, die nach der Rückkehr der Funktion die Unterschnittpaare enthalten. Dieses Array muß mindestens so viele Strukturen enthalten, wie im Parameter *cPairs* angegeben ist. Hat dieser Parameter den Wert *NULL*, gibt die Funktion die insgesamt vorhandene Anzahl der Unterschnittpaare in dieser Schrift zurück.

Rückgabewert Der Rückgabewert bestimmt die Anzahl der Unterschnittpaare, die abgerufen wurden, oder die insgesamt vorhandene Anzahl von Unterschnittpaaren in der Schrift. Der Rückgabewert ist Null, wenn die Funktion fehlschlägt oder keine Unterschnittpaare für diese Schrift existieren.

GetMessageExtraInfo 3.1

Syntax `LONG GetMessageExtraInfo(VOID)`

Beschreibung Die Funktion *GetMessageExtraInfo* ruft die zusätzlichen Informationen der zuletzt durch *GetMessage* oder *PeekMessage* abgerufenen Botschaft ab. Diese zusätzlichen Informationen können einer Botschaft durch den Gerätetreiber der Tastatur oder eines Zeigegeräts hinzugefügt werden.

Parameter Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert enthält die zusätzlichen Informationen, wenn die Funktion erfolgreich ausgeführt wurde. Die Bedeutung der zusätzlichen Informationen ist geräteabhängig.

Siehe auch *GetMessage, hardware_event, PeekMessage*

GetMsgProc 3.1

Syntax

```
LRESULT CALLBACK GetMsgProc(code, wParam, lParam)

int code;        /* Botschaftbearbeitungs-Flag        */
WPARAM wparam;   /* nicht definiert                   */
LPARAM lparam;   /* Zeiger auf eine MSG-Struktur      */
```

Beschreibung Die Funktion *GetMsgProc* ist eine bibliotheksdefinierte Callback-Funktion, die vom System immer dann aufgerufen wird, wenn die Funktion *GetMessage* eine Botschaft aus der Anwendungswarteschlange abgerufen hat. Das System übergibt die abgerufene Botschaft an die Callback-Funktion, bevor die Botschaft an die Fensterfunktion des Zielfensters übergeben wird.

Parameter *code:* Bestimmt, ob die Callback-Funktion die Botschaft bearbeiten oder die Funktion *CallNextHookEx* aufrufen soll. Wenn dieser Parameter kleiner als Null ist, sollte die Callback-Funktion die Botschaft ohne weitere Bearbeitung an *CallNextHookEx* übergeben.

wParam: Enthält einen *NULL*-Wert.

lParam: Zeigt auf eine *MSG*-Struktur, die Informationen über die Botschaft enthält. Die *MSG*-Struktur hat folgende Form:

```
typedef struct tagMSG {    /* msg */
    HWND hwnd;
    UINT message;
    WPARAM wParam;
    LPARAM lParam;
    DWORD time;
    POINT pt;
} MSG;
```

Rückgabewert Die Callback-Funktion sollte Null zurückgeben.

Hinweis Die Callback-Funktion *GetMsgProc* kann die Botschaft wie gewünscht bearbeiten oder untersuchen. Nachdem die Callback-Funktion die Steuerung an das System zurückgegeben hat, gibt die Funktion *GetMessage* die möglicherweise veränderte Botschaft an die Anwendung zurück, die ursprünglich *GetMsgProc* aufgerufen hat. Die Callback-Funktion benötigt keinen Rückgabewert.

Diese Callback-Funktion muß sich in einer dynamischen Linkbibliothek (DLL) befinden.

Anwendungen müssen die Callback-Funktion installieren, indem sie den Filtertyp *WH_GETMESSAGE* und die Adresse der Prozedurinstanz der Callback-Funktion in einem Aufruf der Funktion *SetWindowsHookEx* angeben.

GetMsgProc ist ein Platzhalter für den bibliotheksdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei (.DEF) der Bibliothek exportiert werden.

Siehe auch *CallNextHookEx*, *GetMessage*, *SetWindowsHookEx*

GetNextDriver 3.1

Syntax

```
HDRVR GetNextDriver(hdrvr, fdwFlag)

HDRVR hdrvr;              /* Handle des installier-
                             baren Treibers        */
DWORD fdwFlag;            /* Such-Flag             */
```

Beschreibung Die Funktion *GetNextDriver* listet Instanzen eines installierbaren Treibers auf.

Parameter *hdrvr:* Bezeichnet den installierbaren Treiber, dessen Instanzen aufgelistet werden sollen. Dieser Wert muß mit der Funktion *OpenDriver* abgerufen werden. Ist der Parameter *NULL*, beginnt die Aufzählung – in Abhängigkeit vom Parameter *fdwFlag* – entweder am Ende oder am Anfang der Liste der installierbaren Treiber.

fdwFlag: Legt fest, ob die Funktion nur das Handle der ersten Instanz eines Treibers oder die Handles der Instanzen in der Reihenfolge des Ladens zurückgeben soll. Dieser Parameter kann einen oder mehrere der folgenden Werte annehmen:

Wert	Bedeutung
GND_FIRST-INSTANCEONLY	Die Funktion gibt ein Handle zurück, daß die erste Instanz eines installierbaren Treibers bezeichnet. Wenn dieses Flag gesetzt ist, listet die Funktion nur die erste Instanz eines installierbaren Treibers auf, unabhängig davon, wie oft dieser Treiber installiert wurde.
GND_FORWARD	Listet alle Instanzen des Treibers auf. Die Verwendung dieses Flag entspricht der Nichtbenutzung von *GND_REVERSE*.
GND_REVERSE	Listet die Instanzen des Treibers in der Reihenfolge auf, in der sie geladen wurden. Jeder nachfolgende Aufruf der Funktion gibt das Handle der nächsten Instanz zurück.

Rückgabewert Der Rückgabewert ist das Instanzen-Handle des installierbaren Treibers, wenn die Funktion erfolgreich ist.

GetOpenClipboardWindow 3.1

Syntax
```
HWND GetOpenClipboardWindow(VOID)
```

Beschreibung Die Funktion *GetOpenClipboardWindow* ruft das Handle des Fensters ab, das die Zwischenablage aktuell geöffnet hat.

Parameter Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert ist das Handle des Fensters, das die Zwischenablage geöffnet hat. Schlägt die Funktion fehl, wird *NULL* zurückgegeben.

Siehe auch *GetClipboardOwner, GetClipboardViewer, OpenClipboard*

GetOpenFileName 3.1

Syntax
```
#include <commdlg.h>
BOOL GetOpenFileName(lpofn)

OPENFILENAME FAR* lpofn; /* Adresse der Struktur mit
                            den Initialisierungsdaten */
```

Beschreibung Die Funktion *GetOpenFileName* erzeugt ein systemdefiniertes Dialogfenster, in dem der Benutzer eine zu öffnende Datei auswählen kann.

Parameter *lpofn:* Zeigt auf eine *OPENFILENAME*-Struktur, die die Initialisierungsinformationen für das Dialogfenster enthält. Nach Rückkehr der Funktion *GetOpenFileName* enthält diese Struktur Informationen über die Datei, die der Anwender ausgewählt hat. Die *OPENFILENAME*-Struktur hat folgende Form:

```
#include <commdlg.h>

typedef struct tagOPENFILENAME {  /* ofn */
    DWORD lStructSize;
    HWND hwndOwner;
    HINSTANCE hInstance;
    LPCSTR lpstrFilter;
    LPSTR lpstrCustomFilter;
    DWORD nMaxCustFilter;
    DWORD nFilterIndex;
    LPSTR lpstrFile;
    DWORD nMaxFile;
    LPSTR lpstrFileTitle;
    DWORD nMaxFileTitle;
    LPCSTR lpstrInitialDir;
    LPCSTR lpstrTitle;
```

```
    DWORD Flags;
    UINT nFileOffset;
    UINT nFileExtension;
    LPCSTR lpstrDefExt;
    LPARAM lCustData;
    UINT (CALLBACK* lpfnHook) (HWND, UINT, WPARAM,
                               LPARAM);
    LPCSTR lpTemplateName;
} OPENFILENAME;
```

Rückgabewert Der Rückgabewert ist ungleich Null, wenn der Anwender eine zu öffnende Datei auswählt. Er ist Null, wenn der Anwender den Schalter *Abbrechen* oder den Befehl *Schließen* im Systemmenü wählt. Er ist auch dann Null, wenn der im Element *lpstrFile* der *OPENFILENAME*-Struktur angegebene Puffer zu klein für den ausgewählten Dateinamen ist.

Fehler Verwenden Sie die Funktion *CommDlgExtendedError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

CDERR_FINDRESFAILURE
CDERR_INITIALIZATION
CDERR_LOADRESFAILURE
CDERR_LOADSTRFAILURE
CDERR_LOCKRESFAILURE
CDERR_MEMALLOCFAILURE
CDERR_MEMLOCKFAILURE
CDERR_NOHINSTANCE
CDERR_NOHOOK
CDERR_NOTEMPLATE
CDERR_STRUCTSIZE
FNERR_BUFFERTOOSMALL
FNERR_INVALIDFILENAME
FNERR_SUBCLASSFAILURE

Hinweis Wenn die durch das Element *lpfnHook* der *OPENFILENAME*-Struktur bezeichnete Filterfunktion die *WM_CTLCOLOR*-Botschaft bearbeitet, muß diese Funktion das Handle des Pinsels zurückgeben, der zum Ausmalen des Hintergrunds des Steuerelements benutzt werden soll.

Beispiel Das folgende Beispiel kopiert Dateifilter-Strings in einen Puffer, initialisiert eine *OPENFILENAME*-Struktur und erzeugt ein *Öffnen*-Dialogfenster. Die Dateifilter-Strings werden in folgender Form in der Ressourcendatei gespeichert:

```
STRINGTABLE
BEGIN
  IDS_FILTERSTRING  "Write Files(*.WRI)|*.wri|"+
    "Word Files(*.DOC|*.doc|"
END
```

Das ersetzbare Zeichen am Ende des String wird benötigt, damit der gesamte String in einzelne Strings aufgeteilt, gleichzeitig aber zusammenhängend im Speicher gehalten werden kann.

```
OPENFILENAME ofn;
char szDirName[256];
char szFile[256], szFileTitle[256];
UINT  i, cbString;
char chReplace;  /* String-Trennzeichen für szFilter */
char szFilter[256];
HFILE hf;

/* Den Namen des Systemverzeichnisses holen
   und in DirName speichern */

GetSystemDirectory(szDirName, sizeof(szDirName));
szFile[0] = '\0';

if ((cbString = LoadString(hinst, IDS_FILTERSTRING,
        szFilter, sizeof(szFilter))) == 0) {
    ErrorHandler();
    return 0L;
}
chReplace = szFilter[cbString - 1];
            /* Platzhalterzeichen herausfiltern */
for (i = 0; szFilter[i] != '\0'; i++) {
    if (szFilter[i] == chReplace)
       szFilter[i] = '\0';
}

/* Allen Elementen der Struktur wird Null zugewiesen. */

memset(&ofn, 0, sizeof(OPENFILENAME));

ofn.lStructSize = sizeof(OPENFILENAME);
ofn.hwndOwner = hwnd;
ofn.lpstrFilter = szFilter;
ofn.nFilterIndex = 1;
ofn.lpstrFile = szFile;
ofn.nMaxFile = sizeof(szFile);
ofn.lpstrFileTitle = szFileTitle;
ofn.MaxFileTitle = sizeof(szFileTitle);
ofn.lpstrInitialDir = szDirName;
ofn.Flags = OFN_SHOWHELP | OFN_PATHMUSTEXIST |
            OFN_FILEMUSTEXIST;

if (GetOpenFileName(&ofn)) {
    hf = _lopen(ofn.lpstrFile, OF_READ);
        .
        . /* Hier können Dateioperationen durchgeführt
        .    werden. */
```

```
}
else
    ErrorHandler();
```

Siehe auch *GetSaveFileName*

GetOutlineTextMetrics 3.1

Syntax

```
WORD GetOutlineTextMetrics(hdc, cbData, lpotm)

HDC hdc;                        /* Handle des
                                   Gerätekontexts       */
UINT cbData;                    /* Größe des Puffers für
                                   die Informationen    */
OUTLINETEXTMETRIC FAR* lpotm; /* Zeiger auf OUTLINE-
                                   TEXTMETRIC-Struktur  */
```

Beschreibung Die Funktion *GetOutlineTextMetrics* ruft metrische Daten von TrueType-Schriften ab.

Parameter *hdc:* Bezeichnet den Gerätekontext.

cbData: Bestimmt die Größe des Puffers in Byte, in dem die Informationen zurückgegeben werden.

lpotm: Zeigt auf eine *OUTLINETEXTMETRIC*-Struktur. Hat dieser Parameter den Wert *NULL*, liefert die Funktion die Größe des Datenpuffers zurück, der für die abgerufenen metrischen Daten benötigt wird. Die *OUTLINETEXTMETRIC*-Struktur hat folgende Form:

```
typedef struct tagOUTLINETEXTMETRIC {
    UINT otmSize;
    TEXTMETRIC otmTextMetrics;
    BYTE otmFiller;
    PANOSE otmPanoseNumber;
    UINT otmfsSelection;
    UINT otmfsType;
    UINT otmsCharSlopeRise;
    UINT otmsCharSlopeRun
    UINT otmItalicAngle;
    UINT otmEMSquare;
    INT otmAscent;
    INT otmDescent;
    UINT otmLineGap;
    UINT otmsXHeight;
    UINT otmsCapEmHeight;
    RECT otmrcFontBox;
    INT otmMacAscent;
    INT otmMacDescent;
    UINT otmMacLineGap;
```

```
    UINT otmusMinimumPPEM;
    POINT otmptSubscriptSize;
    POINT otmptSubscriptOffset;
    POINT otmptSuperscriptSize;
    POINT otmptSuperscriptOffset;
    UINT otmsStrikeoutSize;
    INT otmsStrikeoutPosition;
    INT otmsUnderscorePosition;
    UINT otmsUnderscoreSize;
    PSTR otmpFamilyName;
    PSTR otmpFaceName;
    PSTR otmpStyleName;
    PSTR otmpFullName;
} OUTLINETEXTMETRIC;
```

Rückgabewert Bei erfolgreicher Ausführung gibt die Funktion einen Wert ungleich Null, andernfalls Null zurück.

Hinweis Die *OUTLINETEXTMETRIC*-Struktur enthält die meisten metrischen Schriftinformationen, die vom TrueType-Format unterstützt werden, einschließlich einer *TEXTMETRIC*-Struktur. Die letzten vier Elemente der *OUTLINETEXTMETRIC*-Struktur sind Zeiger auf Strings. Anwendungen sollten den für diese Strings benötigten Speicherplatz zusätzlich zu dem für die anderen Elemente benötigten reservieren. Da es keine Systembeschränkungen für die Länge der Strings gibt, sollte die benötigte Größe ermittelt werden, indem der Parameter *lpotm* beim ersten Aufruf der Funktion *GetOutlineTextMetrics* auf *NULL* gesetzt wird.

Siehe auch *GetTextMetrics*

GetQueueStatus 3.1

Syntax

```
DWORD GetQueueStatus(fuFlags)

UINT fuFlags;   /* Status-Flags der Warteschlange     */
```

Beschreibung Die Funktion *GetQueueStatus* gibt einen Wert zurück, der den Typ der Botschaften in der Warteschlange anzeigt.

Diese Funktion ist sehr schnell und wird normalerweise in zeitkritischen Schleifen verwendet, um zu ermitteln, ob die Funktionen *GetMessage* oder *PeekMessage* aufgerufen werden sollten, um Eingaben zu bearbeiten.

GetQueueStatus gibt zwei Arten von Informationen zurück: ob sich neue Botschaften in der Warteschlange befinden, seitdem *GetQueueStatus*, *GetMessage* oder *PeekMessage* das letzte Mal aufgerufen wurden und den Typ der Ereignisse, die sich aktuell in der Warteschlange befinden.

Parameter *fuFlags:* Bestimmt die aus der Warteschlange abzurufenden Informationen. Dieser Parameter kann eine Kombination der folgenden Werte sein:

Wert	Bedeutung
QS_KEY	Eine *WM_CHAR*-Botschaft befindet sich in der Warteschlange.
QS_MOUSE	Eine *WM_MOUSEMOVE*- oder *WM_***BUTTON**-Botschaft befindet sich in der Warteschlange.
QS_MOUSEMOVE	Eine *WM_MOUSEMOVE*-Botschaft befindet sich in der Warteschlange.
QS_MOUSEBOTTON	Eine *WM_***BUTTON**-Botschaft befindet sich in der Warteschlange.
QS_PAINT	Eine *WM_PAINT*-Botschaft befindet sich in der Warteschlange.
QS_POSTMSG	Eine andere Botschaft als die bereits beschriebenen befindet sich in der Warteschlange.
QS_SENDMSG	Die in der Warteschlange befindliche Botschaft stammt von einer anderen Anwendung.
QS_TIMER	Eine *WM_TIMER*-Botschaft befindet sich in der Warteschlange.

Rückgabewert Das höherwertige *WORD* des Rückgabewerts bestimmt den Typ der Botschaften, die sich aktuell in der Warteschlange befinden. Das niederwertige *WORD* bestimmt die Typen der Botschaften, die seit dem letzten Aufruf der Funktionen *GetQueueStatus*, *GetMessage* oder *PeekMessage* in die Warteschlange gestellt wurden und sich noch dort befinden.

Hinweis Die Existenz eines *QS_-Flags* im Rückgabewert garantiert nicht, daß ein nachfolgender Aufruf der Funktionen *PeekMessage* oder *GetMessage* eine Botschaft zurückgibt. *GetMessage* und *PeekMessage* führen interne Filterungen durch, die eine interne Botschaftsbearbeitung veranlassen können. Der Rückgabewert von *GetQueueStatus* ist also nur ein Hinweis darauf, ob *GetMessage* oder *PeekMessage* aufgerufen werden sollte.

Siehe auch *GetInputState*, *GetMessage*, *PeekMessage*

GetRasterizerCaps 3.1

Syntax `BOOL GetRasterizerCaps(lpraststat, cb)`

```
RASTERIZER_STATUS FAR* lprastStat; /* Adresse der
                                      Struktur für
                                      den Status     */
int cb;                           /* Anzahl der Byte
                                      in der Struktur */
```

Beschreibung Die Funktion *GetRasterizerCaps* gibt Flags zurück, die die Installation von TrueType-Schriften im System anzeigen.

Parameter *lpraststat:* Zeigt auf eine *RASTERIZER_STATUS*-Struktur, die Informationen über die Rasterung aufnimmt.

Die *RASTERIZER_STATUS*-Struktur hat folgende Form:

```
typedef struct  tagRASTERIZER_STATUS {    /* rs */
    int nSize;
    int wFlags;
    int nLanguageID;
} RASTERIZER_STATUS;
```

cb: Bestimmt die Anzahl der Byte, die in die durch den Parameter *lpraststat* bezeichnete Struktur kopiert werden.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich ist, andernfalls Null.

Hinweis Mit Hilfe der Funktion *GetRasterizerCaps* können Anwendungen und Druckertreiber ermitteln, ob TrueType installiert ist.

Ist das Flag *TT_AVAILABLE* im Element *wFlags* der *RASTERIZER_STATUS*-Struktur gesetzt, ist mindestens eine TrueType-Schrift installiert.

Wenn das Flag *TT_ENABLED* gesetzt ist, wurde TrueType für das gesamte System aktiviert.

Siehe auch *GetOutlineTextMetrics*

GetSaveFileName 3.1

Syntax

```
#include <commdlg.h>
BOOL GetSaveFileName(lpofn)

OPENFILENAME FAR* lpofn; /* Adresse der Struktur mit
                            den Initialisierungsdaten */
```

Beschreibung Die Funktion *GetSaveFileName* erzeugt ein systemdefiniertes Dialogfenster, in dem der Anwender einen Dateinamen zum Speichern auswählen kann.

Parameter *lpofn:* Zeigt auf eine *OPENFILENAME*-Struktur, die die Initialisierungsinformationen für das Dialogfenster enthält. Nach Rückkehr der Funktion *GetSaveFileName* enthält diese Struktur Informationen über die Dateiauswahl des Anwenders.

Die *OPENFILENAME*-Struktur hat folgende Form:

```
#include <commdlg.h>

typedef struct tagOPENFILENAME {  /* ofn */
    DWORD lStructSize;
    HWND hwndOwner;
    HINSTANCE hInstance;
    LPCSTR lpstrFilter;
    LPSTR lpstrCustomFilter;
    DWORD nMaxCustFilter;
    DWORD nFilterIndex;
    LPSTR lpstrFile;
    DWORD nMaxFile;
    LPSTR lpstrFileTitle;
    DWORD nMaxFileTitle;
    LPCSTR lpstrInitialDir;
    LPCSTR lpstrTitle;
    DWORD Flags;
    UINT nFileOffset;
    UINT nFileExtension;
    LPCSTR lpstrDefExt;
    LPARAM lCustData;
    UINT (CALLBACK* lpfnHook) (HWND, UINT, WPARAM,
                               LPARAM);
    LPCSTR lpTemplateName;
} OPENFILENAME;
```

Rückgabewert Der Rückgabewert ist ungleich Null, wenn der Anwender eine zu sichernde Datei auswählt. Er ist Null, wenn der Anwender den Schalter *Abbrechen* oder den Befehl *Schließen* im Systemmenü wählt. Er ist auch dann Null, wenn der Puffer, der im Element *lpstrFile* der *OPENFILENAME*-Struktur angegeben ist, zu klein für den ausgewählten Dateinamen ist.

Fehler Verwenden Sie die Funktion *CommDlgExtendedError*, um den Fehlerwert zu ermitteln. Möglich sind folgende Werte:

CDERR_FINDRESFAILURE
CDERR_INITIALIZATION
CDERR_LOADRESFAILURE
CDERR_LOADSTRFAILURE
CDERR_LOCKRESFAILURE
CDERR_MEMALLOCFAILURE
CDERR_MEMLOCKFAILURE
CDERR_NOHINSTANCE
CDERR_NOHOOK
CDERR_NOTEMPLATE
CDERR_STRUCTSIZE
FNERR_BUFFERTOOSMALL
FNERR_INVALIDFILENAME
FNERR_SUBCLASSFAILURE

Hinweis

Wenn die durch das Element *lpfnHook* der *OPENFILENAME*-Struktur bezeichnete Filterfunktion die *WM_CTLCOLOR*-Botschaft bearbeitet, muß diese Funktion das Handle des Pinsels zurückgeben, der zum Ausmalen des Hintergrunds des Steuerelements benutzt werden soll.

Beispiel

Das folgende Beispiel kopiert Dateifilterstrings in einen Puffer, initialisiert eine *OPENFILENAME*-Struktur und erzeugt ein Dialogfenster *Speichern als*. Die Dateifilterstrings werden in folgender Form in der Ressourcendatei gespeichert:

```
STRINGTABLE
BEGIN
  IDS_FILTERSTRING  "Write Files(*.WRI)|*.wri|"+
    "Word Files(*.DOC|*.doc|"
END
```

Das ersetzbare Zeichen am Ende des String wird benötigt, damit der gesamte String in einzelne Strings aufgeteilt, gleichzeitig aber zusammenhängend im Speicher gehalten werden kann.

```
OPENFILENAME ofn;
char  szDIRName[256];
char  szFile[256], szFileTitle[256];
UINT  i, cbString;
char  chReplace;  /* String-Trennzeichen für szFilter */
char  szFilter[256];
HFILE hf;

/*
 * Den Namen des Systemverzeichnisses ermitteln und in
 * szDirName speichern.
 */
GetSystemDirectory(szDirName, sizeof(szDirName));

if ((cbString = LoadString(hinst, IDS_FILTERSTRING,
        szFilter, sizeof(szFilter))) == 0) {
    ErrorHandler();
    return 0L;
}

chReplace = szFilter[cbString - 1];
            /* Platzhalterzeichen herausfiltern */

for (i = 0; szFilter[i] != '\0'; i++) {
    if (szFilter[i] == chReplace)
       szFilter[i] = '\0';
}

/* Allen Elementen der Struktur wird Null zugewiesen */

memset(&ofn, 0, sizeof(OPENFILENAME));
```

```
/* OPENFILENAME-Elemente initialisieren */

szFile[0] = '\0';

ofn.lStructSize = sizeof(OPENFILENAME);
ofn.hwndOwner = hwnd;
ofn.lpstrFilter = szFilter;
ofn.lpstrFile = szFile;
ofn.nMaxFile = sizeof(szFile);
ofn.lpstrFileTitle = szFileTitle;
ofn.nMaxFileTitle = sizeof(szFileTitle);
ofn.lpstrInitialDir = szDirName;
ofn.Flags = OFN_SHOWHELP | OFN_OVERWRITEPROMPT;

if (GetSaveFileName(&ofn)) {
    .
    . /* Hier können Dateioperationen durchgeführt
    .    werden */
    .
}
else
    ErrorHandler();
```

Siehe auch *GetOpenFileName*

GetSelectorBase 3.1

Syntax

```
DWORD GetSelectorBase(uSelector)

UINT uSelector;              /* Selektor          */
```

Beschreibung Die Funktion *GetSelectorBase* ruft die Basisadresse eines Selektors ab.

Parameter *uSelector:* Bestimmt den Selektor, dessen Basisadresse abgerufen wird.

Rückgabewert Diese Funktion gibt die Basisadresse des angegebenen Selektors zurück.

Siehe auch *GetSelectorLimit, SetSelectorBase, SetSelectorLimit*

GetSelectorLimit 3.1

Syntax

```
DWORD GetSelectorLimit(uSelector)

UINT Selector;
```

Beschreibung Die Funktion *GetSelectorLimit* ruft die Grenze eines Selektors ab.

Parameter *uSelector:* Bestimmt den Selektor, dessen Grenze abgerufen werden soll.

Rückgabewert Diese Funktion gibt die Grenze des angegebenen Selektors zurück.

Siehe auch *GetSelectorBase*, *SetSelectorBase*, *SetSelectorLimit*

GetSystemDebugState 3.1

Syntax
```
LONG GetSystemDebugState(void)
```

Beschreibung Die Funktion *GetSystemDebugState* ruft Informationen über den Status des Systems ab. Ein Windows-Debugger kann diese Informationen benutzen, um entweder in den Hard-Modus oder in den Soft-Modus umzuschalten, wenn ein Breakpoint angetroffen wird.

Parameter Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert kann einer oder eine Kombination der folgenden Werte sein:

Wert	Bedeutung
SDS_MENU	Das Menü wird angezeigt.
SDS_SYSMODAL	Ein system-modales Dialogfenster wird angezeigt.
SDS_NOTASK-QUEUE	Da die Anwendungswarteschlange noch nicht existiert, kann die Anwendung keine Botschaften entgegennehmen.
SDS_DIALOG	Ein Dialogfenster wird angezeigt.
SDS_TASKIS-LOCKED	Die aktuelle Task ist gesperrt. Aus diesem Grund kann keine andere Task laufen.

GetSystemDir 3.1

Syntax
```
#include <ver.h>
UINT GetSystemDir(lpszWinDir, lpszBuf, cbBuf)

LPCSTR lpszWinDir;   /* Adresse des Windows-
                        Verzeichnisses          */
LPCSTR lpszBuf;      /* Adresse des Puffers für
                        den Pfad                */
int cbBuf;           /* Größe des Puffers       */
```

Beschreibung Diese Funktion ruft den Pfadnamen des Windows-Systemverzeichnisses ab. Dieses Verzeichnis enthält z.B. Windows-Bibliotheken, -Treiber und -Schriftdateien.

GetSystemDir wird von DOS-Programmen aufgerufen, die Windows-Anwendungen installieren. Sie existiert ausschließlich in

der Version für statisches Linken in der Bibliothek für Dateiinstallation. Windows-Anwendungen sollten die Funktion *GetSystemDirectory* verwenden, um das Windows-Systemverzeichnis zu ermitteln.

Parameter

lpszWinDir: Zeigt auf das Windows-Verzeichnis, das durch einen vorhergehenden Aufruf der Funktion *GetWindowsDir* abgerufen wurde.

lpszBuf: Zeigt auf den Puffer, der den nullterminierten String mit dem Pfadnamen aufnehmen soll.

cbBuf: Bestimmt die Größe des durch *lpszBuf* bezeichneten Puffers in Byte.

Rückgabewert

Bei erfolgreicher Ausführung der Funktion ist der Rückgabewert die Länge des in den Puffer kopierten String in Byte. Das abschließende Nullzeichen wird mitgezählt. Ist der Rückgabewert größer als *cbBuf*, gibt er die benötigte Größe des Puffers an, die zum Speichern des Pfadnamens erforderlich ist. Der Rückgabewert ist Null, wenn die Funktion fehlschlägt.

Hinweis

Anwendungen müssen die Funktion *GetWindowsDir* vor *GetSystemDir* aufrufen, um den richtigen Wert für den Parameter *lpszWinDir* zu erhalten.

Der durch diese Funktion abgerufene Pfadname endet nur dann mit einem Backslash (\), wenn das Windows-Systemverzeichnis das Hauptverzeichnis des Datenträgers ist. Heißt das Systemverzeichnis WINDOWS\SYSTEM auf Laufwerk C, lautet der abgerufene Pfadname »C:\WINDOWS\SYSTEM«.

Siehe auch

GetSystemDirectory, GetWindowsDir

GetTextExtentPoint 3.1

Syntax

```
BOOL GetTextExtentPoint(hdc, lpszString, cbString,
   lpSize)

HDC hdc;              /* Handle des Gerätekontexts   */
LPCSTR lpszString;    /* Adresse des Text-String     */
int cbstring;         /* Anzahl Byte im String       */
SIZE FAR* lpSize;     /* Adresse der Struktur für
                         die String-Größe             */
```

Beschreibung

Die Funktion *GetTextExtentPoint* berechnet die Breite und Höhe des angegebenen Text-String. Die Funktion *GetTextExtentPoint* verwendet die aktuelle Schrift zur Berechnung der Abmessungen des String. Die Breite und Höhe wird in virtuellen Einheiten, ohne Berücksichtigung eines möglichen Clipping, berechnet.

Die Funktion *GetTextExtentPoint* kann entweder als Wide-Character-Funktion oder als ANSI-Funktion benutzt werden.

Im ersten Fall müssen die Textargumente in Unicode angegeben werden, im zweiten müssen die Textargumente Zeichen aus dem Zeichensatz von Windows 3.X verwenden.

Parameter *hdc:* Bezeichnet den Gerätekontext.

lpszString: Zeigt auf einen Text-String.

cbString: Bestimmt die Anzahl der Byte im Text-String.

lpSize: Zeigt auf eine *SIZE*-Struktur, die die Abmessungen des String aufnimmt. Die *SIZE*-Struktur hat folgende Form:

```
typedef struct tagSIZE {
    int cx;
    int cy;
} SIZE;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Da einige Geräte Zeichen nicht in regulären Zeichenzellen positionieren – sie unterschneiden –, kann die Summe der Zeichenabmessungen in einem String von den Abmessungen des String selbst abweichen.

Die berechnete Breite berücksichtigt den Zeichenzwischenraum, der mit der Funktion *SetTextCharacterExtra* gesetzt wurde.

Siehe auch *SetTextCharacterExtra*

GetTimerResolution 3.1

Syntax
```
DWORD GetTimerResolution(void)
```

Beschreibung Die Funktion *GetTimerResolution* ruft die Anzahl der Millisekunden je Zeittakt ab.

Parameter Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert ist die Anzahl der Millisekunden je Zeittakt.

Siehe auch *GetTickCount, SetTimer*

GetViewportExtEx 3.1

Syntax
```
BOOL GetViewportExtEx(hdc, lpSize)

HDC hdc;            /* Handle des Gerätekontexts     */
SIZE FAR* lpSize;   /* Zeiger auf SIZE-Struktur      */
```

Beschreibung Die Funktion *GetViewportExtEx* ruft die x- und y-Abmessungen des Grafikfensters des angegebenen Gerätekontexts ab.

Parameter *hdc:* Bezeichnet den Gerätekontext.

lpSize: Zeigt auf eine *SIZE*-Struktur. Die x- und y-Abmessungen werden in dieser Struktur in Geräteeinheiten abgelegt.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

GetViewportOrgEx 3.1

Syntax

```
BOOL GetViewportOrgEx(hdc, lpPoint)

HDC hdc;              /* Handle des Gerätekontexts    */
POINT FAR* lpPoint;   /* Zeiger auf POINT-Struktur    */
```

Beschreibung Die Funktion *GetViewportOrgEx* ruft die x- und y-Koordinaten des Ursprungs des Grafikfensters ab, das dem angegebenen Gerätekontext zugeordnet ist.

Parameter *hdc:* Bezeichnet den Gerätekontext.

lpPoint: Zeigt auf eine *POINT*-Struktur. Der Ursprung des Grafikfensters wird in Gerätekoordinaten in dieser Struktur abgelegt.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

GetWinDebugInfo 3.1

Syntax

```
BOOL GetWinDebugInfo(lpwdi, flags)

WINDEBUGINFO FAR* lpwdi; /* Adresse der WINDEBUGINFO-
                            Struktur                 */
UINT flags;             /* Flag, das die zurückzu-
                            gebenden Informationen
                            bestimmt                 */
```

Beschreibung Die Funktion *GetWinDebugInfo* ruft die aktuellen Debug-Informationen des Systems für die Debug-Version von Windows 3.1 ab.

Parameter *lpwdi:* Zeigt auf eine *WINDEBUGINFO*-Struktur, die mit den Debug-Informationen gefüllt wird. Die *WINDEBUGINFO*-Struktur hat folgende Form:

```
typedef struct tagWINDEBUGINFO {
      UINT flags;
      DWORD dwOptions;
      DWORD dwFilter;
      char achAllocModule[8];
      DWORD dwAllocBreak;
      DWORD dwAllocCount;
} WINDEBUGINFO;
```

flags: Bestimmt, welche Elemente der *WINDEBUGINFO*-Struktur gefüllt werden sollen. Dieser Parameter kann einen oder mehrere der folgenden Werte annehmen:

Wert	Bedeutung
WDI_OPTIONS	Füllt das Element *dwOptions* der *WINDEBUGINFO*-Struktur.
WDI_FILTER	Füllt das Element *dwFilter* der *WINDEBUGINFO*-Struktur.
WDI_ALLOCBREAK	Füllt die Elemente *achAllocModule*, *dwAllocBreak* und *dwAllocCount* der *WINDEBUGINFO*-Struktur.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null. Er ist Null, wenn der durch den Parameter *lpwdi* angegebene Zeiger ungültig ist, oder wenn die Funktion nicht in der Debug-Version von Windows 3.1 aufgerufen wird.

Hinweis Das Element *flags* der zurückgegebenen *WINDEBUGINFO*-Struktur wird auf den Wert gesetzt, der im Parameter *flags* dieser Funktion bereitgestellt wird.

Siehe auch *SetWinDebugInfo*

GetWindowExtEx 3.1

Syntax

```
BOOL GetWindowExtEx(hdc, lpSize)

HDC hdc;                /* Handle des Gerätekontexts */
SIZE FAR* lpSize;       /* Zeiger auf SIZE-Struktur  */
```

Beschreibung Die Funktion *GetWindowExtEx* ruft die x- und y-Abmessungen des Fensters ab, das dem angegebenen Gerätekontext zugeordnet ist.

Parameter *hdc:* Bezeichnet den Gerätekontext.

lpSize: Zeigt auf eine *SIZE*-Struktur. Die x- und y-Abmessungen werden in virtuellen Einheiten in dieser Struktur abgelegt.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

GetWindowOrgEx 3.1

Syntax

```
BOOL GetWindowOrgEx(hdc, lpPoint)

HDC hdc;                /* Handle des Gerätekontexts */
POINT FAR* lpPoint;     /* Zeiger auf POINT-Struktur */
```

Beschreibung Diese Funktion ruft die x- und y-Koordinaten des Ursprungs des Fensters ab, das dem angegebenen Gerätekontext zugeordnet ist.

Parameter *hdc:* Bezeichnet den Gerätekontext.

lpPoint: Zeigt auf eine *POINT*-Struktur. Der Ursprung des Fensters wird in virtuellen Koordinaten in dieser Struktur abgelegt.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

GetWindowPlacement 3.1

Syntax
```
BOOL GetWindowPlacement(hwnd, lpwndpl)

HWND hwnd;                    /* Handle des Fensters  */
WINDOWPLACEMENT FAR* lpwndpl; /* Adresse der Struktur
                                 für die Positions-
                                 daten                */
```

Beschreibung Die Funktion *GetWindowPlacement* ruft den Darstellungsstatus und die normale (wiederhergestellte), die verkleinerte und die vergrößerte Position eines Fensters ab.

Parameter *hwnd:* Bezeichnet das Fenster.

lpwndpl: Zeigt auf die *WINDOWPLACEMENT*-Struktur, die den Darstellungsstatus und die Positionsinformationen aufnimmt. Die *WINDOWPLACEMENT*-Struktur hat folgende Form:

```
typedef struct tagWINDOWPLACEMENT {      /* wndpl */
      UINT length;
      UINT flags;
      UINT showCmd;
      POINT ptMinPosition;
      POINT ptMaxPosition;
      RECT rcNormalPosition;
} WINDOWPLACEMENT;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Siehe auch *SetWindowPlacement*

GetWindowsDir 3.1

Syntax
```
#include <ver.h>
UINT GetWindowsDir(lpszAppDir, lpszPath, cbPath)

LPCSTR lpszAppDir;         /* Adresse des Windows-
                              Verzeichnisses         */
```

```
LPSTR lpszPath;          /* Adresse des Puffers
                            für den Pfad          */
int cbPath;              /* Größe des Puffers
                            für den Pfad          */
```

Beschreibung Die Funktion *GetWindowsDir* ermittelt den Pfadnamen des Windows-Verzeichnisses.

GetWindowsDir wird von DOS-Programmen aufgerufen, die Windows-Anwendungen installieren. Sie existiert ausschließlich in der Version für statisches Linken in der Bibliothek für Dateiinstallation. Windows-Anwendungen sollten die Funktion *GetWindowsDirectory* verwenden, um das Windows-Verzeichnis zu ermitteln.

Parameter *lpszAppDir*: Bestimmt das aktuelle Verzeichnis für eine Suche nach Windows-Dateien. Befindet sich das Windows-Verzeichnis nicht im DOS-Pfad, muß die Anwendung dessen Lage abfragen und den String im Parameter *lpszAppDir* an die Funktion *GetWindowsDir* übergeben.

lpszPath: Zeigt auf den Puffer, der den nullterminierten String mit dem Pfadnamen aufnehmen soll.

cbPath: Bestimmt die Größe des Puffers in Byte, auf den *lpszPath* zeigt.

Rückgabewert Bei erfolgreicher Ausführung ist der Rückgabewert die Länge des String, der in *lpszPath* kopiert wurde. Das abschließende Nullzeichen wird mitgezählt. Wenn der Rückgabewert größer als *cbPath* ist, entspricht er der Größe des Puffers, die erforderlich ist, um den Pfadnamen aufzunehmen. Der Rückgabewert ist Null, wenn die Funktion nicht erfolgreich durchgeführt werden konnte.

Hinweis Der durch diese Funktion abgerufene Pfadname endet nur dann mit einem Backslash (\), wenn das Windows-Verzeichnis das Hauptverzeichnis des Datenträgers ist. Heißt das Windows-Verzeichnis WINDOWS auf Laufwerk C, lautet der abgerufene Pfadname »C:\WINDOWS«. Wurde Windows im Hauptverzeichnis des Laufwerks C installiert, lautet der Pfadname »C:\«.

Nachdem die Funktion *GetWindowsDir* das Windows-Verzeichnis aufgefunden hat, speichert sie dessen Position für nachfolgende Aufrufe der Funktion.

Siehe auch *GetSystemDir, GetWindowsDirectory*

GetWinMem32Version 3.0

Syntax

```
#include <winmem32.h>
WORD GetWinMem32Version(void)
```

Beschreibung Die Funktion *GetWinMem32Version* ruft die Versionsnummer der API ab, die von der dynamischen Linkbibliothek WINMEM32.DLL implementiert wird. Diese Nummer ist nicht die Versionsnummer der Bibliothek selbst.

Parameter Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert bestimmt die Version der 32-Bit-Speicher-API, die durch WINMEM32.DLL implementiert wird. Die höherwertigen 8 Bits enthalten die Hauptversionsnummer, die niederwertigen 8 Bits die Nebenversionsnummer.

Global16PointerAlloc 3.0

Syntax

```
#include <winmem32.h>
WORD Global16PointerAlloc(wSelector, dwOffset,
   lpBuffer, dwSize, wFlags)

WORD wSelector;    /* Selektor des Objekts            */
DWORD dwOffset;    /* Offset des ersten Byte für
                      den Alias                       */
LPDWORD lpBuffer;  /* Adresse der Position des Alias  */
DWORD dwSize;      /* Größe des Bereichs              */
WORD wFlags;       /* reserviert, muß Null sein       */
```

Beschreibung Die Funktion *Global16PointerAlloc* konvertiert einen 16:32-Zeiger in ein 16:16-Zeigeralias, den die Anwendung an eine Windows-Funktion oder an andere 16:16-Funktionen übergeben kann.

Parameter *wSelector:* Bestimmt den Selektor des Objekts, für das ein Alias erzeugt werden soll. Der Selektor muß durch einen vorhergehenden Aufruf der Funktion *Global32Alloc* zurückgegeben worden sein.

dwOffset: Bestimmt den Offset des ersten Byte, für das ein Alias erzeugt werden soll. Der Offset ist ausgehend vom ersten Byte des Objekts gebildet, das der Parameter *wSelector* angibt. Beachten Sie, daß *wSelector:dwOffset* eine 16:32-Adresse des ersten Byte des Bereichs bildet, für den ein Alias erzeugt werden soll.

lpBuffer: Zeigt auf eine Vier-Byte-Position im Speicher, die den 16:16-Zeigeralias für den angegebenen Bereich aufnimmt.

dwSize: Bestimmt die adressierbare Größe des Bereichs in Byte, für den ein Alias erzeugt werden soll. Dieser Wert darf nicht größer als 64 KByte sein.

wFlags: Reserviert, muß Null sein.

Rückgabewert Der Rückgabewert ist Null, wenn die Funktion erfolgreich ist. Andernfalls handelt es sich um einen Fehlerwert. Folgende Werte sind möglich:

WM32_Insufficient_Mem
WM32_Insufficient_Sels
WM32_Invalid_Arg
WM32_Invalid_Flags
WM32_Invalid_Func

Hinweis

Wenn die Funktion erfolgreich ausgeführt werden konnte, enthält die durch den Parameter *lpBuffer* bezeichnete Position einen 16:16-Zeiger auf das erste Byte des Bereichs. Dieses Byte wird auch durch *wSelector:dwOffset* bezeichnet.

Der zurückgegebene Selektor bezeichnet einen Deskriptor eines Datensegments mit den folgenden Attributen: Lesen-Schreiben, Nach oben erweitern und klein, d.h. mit gelöschtem B-Bit. Die Berechtigungsstufe des Deskriptors (DPL) und die Granularität (das G-Bit) wird vom System festgelegt. Sie sollten daher nicht von bestimmten Werten ausgehen. Die Berechtigungsstufen des Deskriptors und des Requestors (DPL und RPL) sind für Windows-Anwendungen geeignet.

Anwendungen dürfen die Einstellungen der Bits im DPL- oder im RPL-Selektor nicht verändern. Änderungen dieser Einstellungen können zum Absturz des Systems führen oder die Lauffähigkeit der Anwendung auf kompatiblen Geräten verhindern.

Wegen der Speicheraufteilung einiger Systeme muß der Offset-Bestandteil des zurückgegebenen 16:16-Zeigers nicht unbedingt Null sein.

Beim Schreiben Ihrer Anwendung sollten Sie nicht von einer bestimmten Größe des zurückgegebenen Selektors ausgehen. Statt dessen sollten Sie unterstellen, daß mindestens *dwSize* Byte – beginnend beim 16:16-Zeiger, der von dieser Funktion erzeugt wird – adressiert werden können.

Siehe auch

Global16PointerFree

Global16PointerFree 3.0

Syntax

```
#include <winmem32.h>
WORD Global16PointerFree(wSelector, dwAlias, wFlags)

WORD wSelector;       /* Selektor des Objekts         */
DWORD dwAlias;        /* freizugebender Zeigeralias   */
WORD wFlags;          /* reserviert, muß Null sein    */
```

Beschreibung

Die Funktion *Global16PointerFree* gibt den 16:16-Zeigeralias frei, der mit einem vorhergehenden Aufruf der Funktion *Global16PointerAlloc* erzeugt wurde.

Parameter

wSelector: Bestimmt den Selektor des Objekts, dessen Alias freigegeben werden soll. Der Selektor muß durch einen vorhergehenden Aufruf der Funktion *Global32Alloc* zurückgegeben worden sein.

dwAlias: Bestimmt den 16:16-Zeigeralias, der freigegeben werden soll. Dieser Alias muß einschließlich des ursprünglichen Offsets von einem vorhergehenden Aufruf der Funktion *Global16PointerAlloc* zurückgegeben worden sein.

wFlags: Reserviert, muß Null sein.

Rückgabewert Der Rückgabewert ist Null, wenn die Funktion erfolgreich ist. Andernfalls handelt es sich um einen Fehlerwert. Folgende Werte sind möglich:

WM32_Insufficient_Mem
WM32_Insufficient_Sels
WM32_Invalid_Arg
WM32_Invalid_Flags
WM32_Invalid_Func

Hinweis Anwendungen sollten einen 16:16-Zeigeralias sofort freigeben, wenn er nicht mehr benötigt wird. Die Freigabe des Alias gibt auch Speicher in der Deskriptor-Tabelle frei. Letztere ist eine knappe Systemressource.

Siehe auch *Global16PointerAlloc*

Global32Alloc 3.0

Syntax

```
#include <winmem32.h>
WORD Global32Alloc(dwSize, lpSelector, dwMaxSize,
   wFlags)

DWORD dwSize;        /* Größe des zu reservierenden
                        Blocks                      */
LPWORD lpSelector;   /* Adresse der Position für
                        den Selektor                */
DWORD dwMaxSize;     /* maximale Größe des Objekts  */
WORD wFlags;         /* für gemeinsamen Zugriff     */
```

Beschreibung Die Funktion *Global32Alloc* reserviert ein Speicherobjekt, das als 16:32-Code- oder Datensegment (USE32) benutzt werden soll. Sie ruft außerdem den Selektoranteil der 16:32-Adresse des Speicherobjekts ab. Das erste Byte des Objekts befindet sich – ausgehend von diesem Selektor – am Offset Null.

Parameter *dwSize:* Bestimmt die anfängliche Größe des zu reservierenden Objekts in Byte. Dieser Wert muß sich im Bereich zwischen 1 und (16 MByte - 64 KByte) befinden.

lpSelector: Zeigt auf eine 2-Byte-Position im Speicher, die den Selektoranteil der 16:32-Adresse des reservierten Objekts aufnimmt.

dwMaxSize: Bestimmt die maximale Größe des Objekts in Byte, wenn es durch die Funktion *Global32Realloc* erneut reserviert

wird. Dieser Wert muß sich im Bereich von 1 bis (16 MByte - 64 KByte) befinden. Wird dieses Speicherobjekt durch die Anwendung nicht erneut zugewiesen, sollte der Parameter *dwMaxSize* auf den gleichen Wert wie der Parameter *dwSize* gesetzt werden.

wFlags: Hängt vom Rückgabewert der Funktion *GetWinMem32Version* ab. Ist der Rückgabewert kleiner als 0x0101, muß dieser Parameter Null sein. Wenn der Rückgabewert größer oder gleich 0x0101 ist, so kann dieser Parameter auf *GMEM_DDESHARE* gesetzt werden, um den gemeinsamen Zugriff auf dieses Objekt zu ermöglichen. Andernfalls sollte dieser Parameter Null sein. Weitere Informationen über *GMEM_DDESHARE* finden Sie in der Beschreibung der Funktion *GlobalAlloc*.

Rückgabewert

Der Rückgabewert ist Null, wenn die Funktion erfolgreich ist. Andernfalls handelt es sich um einen Fehlerwert. Folgende Werte sind möglich:

WM32_Insufficient_Mem
WM32_Insufficient_Sels
WM32_Invalid_Arg
WM32_Invalid_Flags
WM32_Invalid_Func

Hinweis

Schlägt die Funktion *Global32Alloc* fehl, ist der durch den Parameter *lpSelector* bezeichnete Wert Null. Ist die Funktion erfolgreich, zeigt *lpSelector* auf den Selektor des Objekts. Der gültige Bereich der Offsets der durch diesen Selektor referenzierten Objekte ist 0 bis *dwSize*.

Unter Windows 3.0 und neueren Versionen ist die maximale Größe für das zuzuweisende Objekt 0x00FF0000 (16 MByte - 64 KByte) groß. Diese Beschränkung für WINMEM32.DLL stammt vom aktuellen Windows-Kernel.

Der zurückgegebene Selektor bezeichnet einen Deskriptor eines Datensegments mit den folgenden Attributen: Lesen-Schreiben, Nach oben erweitern und klein, d.h. mit gelöschtem B-Bit. Die Berechtigungsstufe des Deskriptors (DPL) und die Granularität (das G-Bit) wird vom System festgelegt. Sie sollten daher nicht von bestimmten Werten ausgehen. Die Berechtigungsstufen des Deskriptors und des Requestors (DPL und RPL) sind für Windows-Anwendungen geeignet.

Der zurückgegebene Selektor bezeichnet einen Deskriptor eines Datensegments mit den folgenden Attributen: Lesen-Schreiben, Nach oben erweitern und groß, d.h. mit gesetztem B-Bit. Die Berechtigungsstufe des Deskriptors (DPL) und die Granularität (das G-Bit) wird vom System festgelegt. Sie sollten daher nicht von bestimmten Werten ausgehen. Da die Größe der Granularität vom System festgelegt wird, kann die Größe

des Objekts und die Beschränkung der Selektor-Größe um bis zu 4095 Byte (4 KByte -1) größer als die angeforderte Größe sein.

Die Berechtigungsstufen des Deskriptors und des Requestors (DPL und RPL) sind für Windows-Anwendungen geeignet.

Anwendungen dürfen die Einstellung der Bits im DPL- oder im RPL-Selektor nicht verändern. Eine Veränderung der Einstellungen kann zu einem Absturz des Systems führen oder die Lauffähigkeit der Anwendung auf kompatiblen Geräten verhindern.

Das reservierte Objekt ist weder verschiebbar noch verwerfbar, kann aber in Speicherseiten aufgeteilt werden. Anwendungen sollten 32-Bit-Speicherobjekte nicht mit einer Seitensperre belegen. Dies wäre auch nur dann hilfreich, wenn das Objekt Code oder Daten enthielte, die von einem Interrupt benötigt werden. Ein 32-Bit-Speicher kann aber von einem Interrupt nicht benutzt werden.

Siehe auch *Global32Free, Global32Realloc*

Global32CodeAlias 3.0

Syntax

```
#include <winmem32.h>
WORD Global32CodeAlias(wSelector, lpAlias, wFlags)

WORD wSelector;    /* Selektor des Objekts des Alias  */
LPWORD lpAlias;    /* Adresse der Position für den
                      Aliasselektor                   */
WORD wFlags;       /* reserviert, muß Null sein       */
```

Beschreibung Die Funktion *Global32CodeAlias* erzeugt einen 16:32-Codesegment-Aliasselektor (USE 32) für ein 32-Bit-Speicherobjekt, das zuvor mit der Funktion *Global32Alloc* erzeugt wurde. Dies ermöglicht es Anwendungen, Code auszuführen, der sich in diesem Speicherobjekt befindet.

Parameter *wSelector:* Bestimmt den Selektor des Objekts, für das ein Alias erzeugt werden soll. Hierbei muß es sich um den Selektor handeln, der von einem vorhergehenden Aufruf der Funktion *Global32Alloc* zurückgegeben wurde.

lpAlias: Zeigt auf eine 2-Byte-Position im Speicher, die den Selektoranteil des 16:32-Codesegment-Alias für das angegebene Objekt aufnimmt.

wFlags: Reserviert, muß Null sein.

Rückgabewert Der Rückgabewert ist Null, wenn die Funktion erfolgreich ist. Andernfalls handelt es sich um einen Fehlerwert. Folgende Werte sind möglich:

WM32_Insufficient_Mem
WM32_Insufficient_Sels
WM32_Invalid_Arg
WM32_Invalid_Flags
WM32_Invalid_Func

Hinweis

Schlägt die Funktion fehl, ist der durch den Parameter *lpAlias* bezeichnete Wert Null. Ist die Funktion erfolgreich, so zeigt *lpAlias* auf ein USE32-Codesegment-Alias für das durch den Parameter *wSelector* angegebene Objekt. Das erste Byte dieses Objekts befindet sich am Offset 0, ausgehend von dem Selektor, der in *lpAlias* zurückgegeben wird. Gültige Offsets ergeben sich aus der Größe des Objekts, die mit dem zuletzt erfolgten Aufruf der Funktion *Global32Alloc* oder *Global32Realloc* gesetzt wurde.

Der zurückgegebene Selektor bezeichnet einen Deskriptor eines Datensegments mit den folgenden Attributen: Lesen-Ausführen, nicht-konform und USE32, d.h. mit gesetztem D-Bit. Die Berechtigungsstufe des Deskriptors (DPL) und die Granularität (das G-Bit) wird vom System festgelegt. Sie sollten daher nicht von bestimmten Werten ausgehen. Die Granularität stimmt mit der des Datenselektors dieses Objekts überein. Die Berechtigungsstufen des Deskriptors und des Requestors (DPL und RPL) sind für Windows-Anwendungen geeignet.

Anwendungen dürfen die Einstellung der Bits im DPL- oder im RPL-Selektor nicht ändern. Eine Änderung dieser Einstellung könnte zum Absturz des Systems führen oder die Lauffähigkeit der Anwendung auf kompatiblen Geräten verhindern.

Anwendungen sollen diese Funktion nur ein einziges Mal für ein Objekt aufrufen. In Abhängigkeit vom System kann die Funktion fehlschlagen, wenn sie von einer Anwendung ein zweites Mal für ein bestimmtes Objekt aufgerufen wird, ohne daß zuvor die Funktion *Global32CodeAliasFree* aufgerufen wurde.

Siehe auch

Global32Alloc, Global32CodeAliasFree

Global32CodeAliasFree 3.0

Syntax

```
#include <winmem32.h>
WORD Global32CodeAliasFree(wSelector, wAlias, wFlags)

WORD wSelecotr;    /* Selektor des Objekts           */
WORD wAlias;       /* freizugebender Codesegment-
                      Aliasselektor                  */
WORD wFlags;       /* reserviert, muß Null sein      */
```

Beschreibung

Die Funktion *Global32CodeAliasFree* gibt den 16:32-Codesegment-Aliasselektor (USE32) frei, der durch einen vorhergehenden Aufruf der Funktion *Global32CodeAlias* erzeugt wurde.

Parameter *wSelector:* Bestimmt den Selektor des Objekts, dessen Alias freigegeben werden soll. Dieser Selektor muß durch einen vorhergehenden Aufruf der Funktion *Global32Alloc* zurückgegeben worden sein.

wAlias: Bestimmt den USE32-Codesegment-Aliasselektor, der freigegeben werden soll. Dies muß der Alias sein, der von einem vorhergehenden Aufruf der Funktion *Global32CodeAlias* zurückgegeben wurde.

wFlags: Reserviert, muß Null sein.

Rückgabewert Der Rückgabewert ist Null, wenn die Funktion erfolgreich ist. Andernfalls handelt es sich um einen Fehlerwert. Folgende Werte sind möglich:

WM32_Insufficient_Mem
WM32_Insufficient_Sels
WM32_Invalid_Arg
WM32_Invalid_Flags
WM32_Invalid_Func

Siehe auch *Global32CodeAlias*

Global32Free 3.0

Syntax

```
#include <winmem32.h>
WORD Global32Free(wSelector, wFlags)

WORD wSelektor;       /* Selektor des freizugebenden
                         Objekts                    */
WORD wFlags;          /* reserviert, muß Null sein  */
```

Beschreibung Die Funktion *Global32Free* gibt ein Objekt frei, das durch einen vorhergehenden Aufruf der Funktion *Global32Alloc* reserviert wurde.

Parameter *wSelector:* Bestimmt den Selektor des freizugebenden Objekts. Dieser Selektor muß von einem vorhergehenden Aufruf der Funktion *Global32Alloc* zurückgegeben worden sein.

wFlags: Reserviert, muß Null sein.

Rückgabewert Der Rückgabewert ist Null, wenn die Funktion erfolgreich ist. Andernfalls handelt es sich um einen Fehlerwert. Möglich sind folgende Werte:

WM32_Insufficient_Mem
WM32_Insufficient_Sels
WM32_Invalid_Arg
WM32_Invalid_Flags
WM32_Invalid_Func

Hinweis

Die Funktion *Global32Alloc* gibt das Objekt selbst frei. Außerdem werden alle Alias freigegeben, die von der 32-Bit-Speicher-API für dieses Objekt erzeugt wurden.

Anmerkung: Vor ihrer Beendigung muß eine Anwendung diese Funktion aufrufen, um jedes Objekt freizugeben, das mit der Funktion *Global32Alloc* reserviert wurde. Nur so wird sichergestellt, daß alle für dieses Objekt erzeugten Alias freigegeben werden.

Siehe auch

Global32Alloc, Global32Realloc

Global32Realloc 3.0

Syntax

```
#include <winmem32.h)
WORD Global32Realloc(wSelector, dwNewSize, wFlags)

WORD wSelector;       /* Selektor des erneut zu
                         reservierenden Objekts      */
DWORD dwNewSize;      /* neue Größe des Objekts      */
WORD wFlags;          /* reserviert, muß Null sein   */
```

Beschreibung

Die Funktion *Global32Realloc* ändert die Größe eines 32-Bit-Speicherobjekts, das zuvor mit der Funktion *Global32Alloc* reserviert wurde.

Parameter

wSelector: Bestimmt den Selektor des Objekts, das geändert werden soll. Dieser Selektor muß von einem vorhergehenden Aufruf der Funktion *Global32Alloc* zurückgegeben worden sein.

dwNewSize: Bestimmt die neue Größe des Objekts in Byte. Dieser Wert muß größer als Null und kleiner oder gleich der Größe sein, die im Parameter *dwMaxSize* der Funktion *Global32Alloc* bei der Erzeugung des Objekts angegeben wurde.

wFlags: Reserviert, muß Null sein.

Rückgabewert

Der Rückgabewert ist Null, wenn die Funktion erfolgreich ist. Andernfalls handelt es sich um einen Fehlerwert. Folgende Werte sind möglich:

WM32_Insufficient_Mem
WM32_Insufficient_Sels
WM32_Invalid_Arg
WM32_Invalid_Flags
WM32_Invalid_Func

Hinweis

Wenn diese Funktion fehlschlägt, bleibt der bisherige Status des Objekts unverändert. Ist die Funktion erfolgreich, wird der Status des Objekts und der Status aller Alias auf dieses Objekt, die durch die 32-Bit-Speicher-API-Funktionen erzeugt wurden, aktualisiert. Aus diesem Grund müssen Anwendungen die

Funktion *Global32Realloc* aufrufen, um die Größe eines Objekts zu ändern. Die Verwendung anderer Windows-Funktionen macht die Alias auf dieses Objekt ungültig.

Diese Funktion ändert den im Parameter *wSelector* angegebenen Selektor nicht. Ist sie erfolgreich, bewegt sich der neue Gültigkeitsbereich der Offsets für diesen Selektor zwischen Null und *dwNewSize*.

Das System legt die geeignete Granularität des Objekts fest. Deshalb kann die Größe des Objekts und die Größenbeschränkung des Selektors um bis zu 4095 Byte (4 KByte - 1) größer sein als die angeforderte Größe.

Siehe auch *Global32Alloc, Global32Free*

GlobalEntryHandle 3.1

Syntax

```
#include <toolhelp.h>
BOOL GlobalEntryHandle(lpge, hglb)

GLOBALENTRY FAR* lpge;   /* Adresse der Struktur für
                            das Objekt               */
HGLOBAL hglb;            /* Handle des Elements      */
```

Beschreibung Die Funktion *GlobalEntryHandle* füllt die angegebene Struktur mit Informationen, die das angegebene globale Speicherobjekt beschreiben.

Parameter *lpge:* Zeigt auf eine *GLOBALENTRY*-Struktur, die die Informationen über das globale Speicherobjekt aufnimmt.

Die *GLOBALENTRY*-Struktur hat folgende Form:

```
#include <toolhelp.h>

typedef struct tagGLOBALENTRY { /* ge */
      DWORD dwSize;
      DWORD dwAddress;
      DWORD dwBlockSize;
      HGLOBAL hBlock;
      WORD wcLock;
      WORD wcPageLock;
      WORD wFlags;
      BOOL wHeapPresent;
      HGLOBAL hOwner;
      WORD wType;
      WORD wData;
      DWORD dwNext;
      DWORD dwNextAlt;
} GLOBALENTRY;
```

hglb: Bezeichnet das globale Speicherobjekt, das beschrieben werden soll.

Rückgabewert Ist die Funktion erfolgreich, ist der Rückgabewert ungleich Null, andernfalls Null. Die Funktion schlägt fehl, wenn der Wert von *hglb* ein ungültiges Handle oder ein ungültiger Selektor ist.

Hinweis Diese Funktion ruft Informationen über ein globales Speicher-Handle oder einen globalen Speicherselektor ab. Debugger können diese Funktion benutzen, um die Segmentnummer eines Segments zu erhalten, das aus einer ausführbaren Datei geladen wurde.

Anwendungen müssen die *GLOBALENTRY*-Struktur initialisieren und ihre Größe im Element *dwSize* in Byte angeben, bevor sie die Funktion *GlobalEntryHandle* aufrufen.

Siehe auch *GlobalEntryModule, GlobalFirst, GlobalInfo, GlobalNext*

GlobalEntryModule 3.1

Syntax

```
#include <toolhelp.h>
BOOL GlobalEntryModule(lpge, hmod, wSeg)

GLOBALENTRY FAR* lpge;   /* Adresse der Struktur für
                            das Segment              */
HMODULE hmod;            /* Handle des Moduls       */
WORD wSeg;               /* zu beschreibendes
                            Segment                  */
```

Beschreibung Die Funktion *GlobalEntryModule* füllt die angegebene Struktur mit Informationen über das angegebene Modulsegment.

Parameter *lpge:* Zeigt auf eine *GLOBALENTRY*-Struktur, die Informationen über das im Parameter *wSeg* angegebene Segment aufnimmt. Die *GLOBALENTRY*-Struktur hat folgende Form:

```
#include <toolhelp.h>

typedef struct tagGLOBALENTRY { /* ge */
      DWORD dwSize;
      DWORD dwAddress;
      DWORD dwBlockSize;
      HGLOBAL hBlock;
      WORD wcLock;
      WORD wcPageLock;
      WORD wFlags;
      BOOL wHeapPresent;
      HGLOBAL hOwner;
      WORD wType;
      WORD wData;
      DWORD dwNext;
      DWORD dwNextAlt;
} GLOBALENTRY;
```

hmod: Bezeichnet das Modul, dem das Segment zugeordnet ist.

wSeg: Bestimmt das Segment, das in der *GLOBALENTRY*-Struktur beschrieben werden soll. Die Nummer des ersten Segments im Modul ist 1. Segmentnummern sind immer fortlaufend. Wenn die letzte gültige Segmentnummer 10 ist, sind alle Segmentnummern zwischen 1 und 10 gültig.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null. Diese Funktion schlägt fehl, wenn das im Parameter *wSeg* angegebene Segment im Modul *hmod* nicht existiert.

Hinweis Debugger können die Funktion *GlobalEntryModul* benutzen, um Informationen über den globalen Heap eines bestimmten Segments abzurufen, das aus einer ausführbaren Datei geladen wurde. Normalerweise arbeitet der Debugger mit Symbolen, die sich auf Segmentnummern beziehen. Diese Funktion übersetzt die Segmentnummern in Heap-Informationen.

Anwendungen müssen die *GLOBALENTRY*-Struktur initialisieren und ihre Größe im Element *dwSize* in Byte angeben, bevor sie *GlobalEntryModule* aufrufen.

Siehe auch *GlobalEntryHandle, GlobalFirst, GlobalInfo, GlobalNext*

GlobalFirst 3.1

Syntax

```
#include <toolhelp.h>
BOOL GlobalFirst(lpge, wFlags)

GLOBALENTRY FAR* lpge;   /* Adresse der Struktur für
                            das Objekt              */
WORD wFlags;             /* Bestimmt den zu
                            benutzenden Heap        */
```

Beschreibung Die Funktion *GlobalFirst* füllt die angegebene Struktur mit Informationen, die das erste Objekt auf dem globalen Heap beschreiben.

Parameter *lpge:* Zeigt auf eine *GLOBALENTRY*-Struktur, die Informationen über das globale Speicherobjekt aufnimmt. Die *GLOBALENTRY*-Struktur hat folgende Form:

```
#include <toolhelp.h>

typedef struct tagGLOBALENTRY { /* ge */
      DWORD dwSize;
      DWORD dwAddress;
      DWORD dwBlockSize;
      HGLOBAL hBlock;
      WORD wcLock;
      WORD wcPageLock;
```

```
    WORD wFlags;
    BOOL wHeapPresent;
    HGLOBAL hOwner;
    WORD wType;
    WORD wData;
    DWORD dwNext;
    DWORD dwNextAlt;
} GLOBALENTRY;
```

wFlags: Bestimmt den zu benutzenden Heap. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
GLOBAL_ALL	Die durch *lpge* bezeichnete Struktur nimmt Informationen über das erste Objekt auf dem globalen Heap auf.
GLOBAL_FREE	Die Struktur nimmt Informationen über das erste Objekt in der Liste der freien Speicherblöcke auf.
GLOBAL_LRU	Die Struktur nimmt Informationen über das erste Objekt in der Liste der am längsten nicht benutzten (LRU, least recently used) Objekte auf.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *GlobalFirst* kann zum Beginn einer Überprüfung des globalen Heap benutzt werden. Anwendungen können die nachfolgenden Objekte auf dem globalen Heap mit Hilfe der Funktion *GlobalNext* untersuchen. Aufrufe von *GlobalNext* müssen mit demselben Wert für *wFlags* erfolgen wie der Aufruf von *GlobalFirst*.

Anwendungen müssen die *GLOBALENTRY*-Struktur initialisieren und ihre Größe im Element *dwSize* in Byte angeben, bevor sie *GlobalFirst* aufrufen.

Siehe auch *GlobalEntryHandle, GlobalEntryModule, GlobalInfo, GlobalNext*

GlobalHandleToSel 3.1

Syntax

```
#include <toolhelp.h>
WORD GlobalHandleToSel(hglb)

HGLOBAL hglb;  /* Handle des globalen Speicherobjekts */
```

Beschreibung Die Funktion *GlobalHandleToSel* konvertiert das angegebene Handle in einen Selektor.

Parameter *hglb:* Bezeichnet das globale Speicherobjekt, das konvertiert werden soll.

Rückgabewert Der Rückgabewert ist der Selektor des angegebenen Objekts, wenn die Funktion erfolgreich war. Andernfalls ist er Null.

Hinweis Die Funktion *GlobalHandleToSel* konvertiert ein globales Handle in einen Selektor, der je nach laufender Version zu Windows 3.0 oder 3.1 paßt. Debugger können diesen Selektor zum Zugriff auf ein globales Speicherobjekt verwenden, wenn das Objekt nicht verwerfbar oder die Attribute des Objekts unerheblich sind.

Siehe auch *GlobalAlloc*

GlobalInfo 3.1

Syntax
```
#include <toolhelp.h>
BOOL GlobalInfo(lpgi)

GLOBALINFO FAR* lpgi; /* Adresse der GLOBALINFO-
                         Struktur                    */
```

Beschreibung Die Funktion *GLOBALINFO* füllt die angegebene Struktur mit Informationen, die den globalen Heap beschreiben.

Parameter *lpgi:* Zeigt auf eine *GLOBALINFO*-Struktur, die Informationen über den globalen Heap aufnimmt. Die *GLOBALINFO*-Struktur hat folgende Form:

```
#include <toolhelp.h>

typedef struct tagGLOBALINFO { /* gi */
      DWORD dwSize;
      WORD wcItems;
      WORD wcItemsFree;
      WORD wcItemsLRU;
} GLOBALINFO;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Informationen in der Struktur können benutzt werden, um den benötigten Speicherplatz für eine Überprüfung des globalen Heap zu ermitteln.

Anwendungen müssen die *GLOBALINFO*-Struktur initialisieren und ihre Größe im Element *dwSize* in Byte angeben, bevor sie die Funktion *GlobalInfo* aufrufen.

Siehe auch *GlobalEntryHandle, GlobalEntryModule, GlobalFirst, GlobalNext*

GlobalNext 3.1

Syntax

```
#include <toolhelp.h>
BOOL GlobalNext(lpge, flags)

GLOBALENTRY FAR* lpge;   /* Adresse der Struktur für
                            das Objekt              */
WORD flags;              /* zu verwendender Heap    */
```

Beschreibung

Die Funktion *GlobalNext* füllt die angegebene Struktur mit Informationen, die das nächste Objekt auf dem globalen Heap beschreiben.

Parameter

lpge: Zeigt auf eine *GLOBALENTRY*-Struktur, die Informationen über das globale Speicherobjekt aufnimmt. Die *GLOBALENTRY*-Struktur hat folgende Form:

```
#include <toolhelp.h>

typedef struct tagGLOBALENTRY { /* ge */
      DWORD dwSize;
      DWORD dwAddress;
      DWORD dwBlockSize;
      HGLOBAL hBlock;
      WORD wcLock;
      WORD wcPageLock;
      WORD wFlags;
      BOOL wHeapPresent;
      HGLOBAL hOwner;
      WORD wType;
      WORD wData;
      DWORD dwNext;
      DWORD dwNextAlt;
} GLOBALENTRY;
```

flags: Bestimmt den zu benutzenden Heap. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
GLOBAL_ALL	Die durch *lpge* bezeichnete Struktur nimmt Informationen über das erste Objekt auf dem globalen Heap auf.
GLOBAL_FREE	Die Struktur nimmt Informationen über das erste Objekt in der Liste der freien Speicherblöcke auf.
GLOBAL_LRU	Die Struktur nimmt Informationen über das erste Objekt in der Liste der am längsten nicht benutzten (LRU, least recently used) Objekte auf.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *GlobalNext* kann verwendet werden, um eine Überprüfung des globalen Heap fortzusetzen, die mit der Funktion *GlobalFirst*, *GlobalEntryHandle* oder *GlobalEntryModule* gestartet wurde.

Wurde die Überprüfung mit *GlobalFirst* gestartet, muß der Parameter *flags* im Aufruf der Funktion *GlobalNext* den gleichen Wert haben wie beim Aufruf von *GlobalFirst*.

Siehe auch *GlobalEntryHandle*, *GlobalEntryModule*, *GlobalFirst*, *GlobalInfo*

GrayStringProc 2.x

Syntax
```
BOOL CALLBACK GrayStringProc(hdc, lpData, cch)

HDDC hdc;           /* Handle des Gerätekontexts    */
LPARAM lpData;      /* Adresse des zu zeichnenden
                       String                       */
int cch;            /* Länge des zu zeichnenden
                       String                       */
```

Beschreibung Die Funktion *GrayStringProc* ist eine anwendungsdefinierte Callback-Funktion, die einen String als Ergebnis eines Aufrufs der Funktion *GrayString* zeichnet.

Parameter *hdc:* Bezeichnet einen Gerätekontext mit einem Bitmap, das mindestens die Breite und die Höhe hat, die in den an die Funktion *GrayString* übergebenen Parametern *cx* und *cy* angegeben war.

lpData: Zeigt auf den zu zeichnenden String.

cch: Bestimmt die Länge des String in Zeichen.

Rückgabewert Wenn die Callback-Funktion erfolgreich ist, sollte sie *TRUE*, andernfalls *FALSE* zurückgeben.

Hinweis Die Callback-Funktion muß den String relativ zu den Koordinaten (0,0) zeichnen.

GrayStringProc ist ein Platzhalter für den anwendungsdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei (.DEF) der Anwendung exportiert werden.

Siehe auch *GrayString*

HardwareProc 3.1

Syntax
```
LRESULT CALLBACK HardwareProc(code, wParam, lParam)

int code;           /* Filtercode                   */
```

```
WPARAM wParam;        /* nicht definiert              */
LPARAM lParam;        /* Adresse der Struktur mit
                         den Ereignisinformationen    */
```

Beschreibung

Die Funktion *HardwareProc* ist eine anwendungsdefinierte Callback-Funktion, die vom System immer dann aufgerufen wird, wenn die Anwendung *GetMessage* oder *PeekMessage* aufruft, weil ein Hardware-Ereignis zu bearbeiten ist. Maus- und Tastaturereignisse werden von diesem Filter nicht bearbeitet.

Parameter

code: Bestimmt, ob die Callback-Funktion die Botschaft bearbeiten oder die Funktion *CallNextHookEx* aufrufen soll. Ist der Wert kleiner als Null, sollte die Callback-Funktion die Botschaft an *CallNextHookEx* weitergeben, ohne sie zu bearbeiten. Ist dieser Wert *HC_NOREMOVE*, benutzt die Anwendung die Funktion *PeekMessage* mit der Option *PM_NOREMOVE*. In diesem Fall wird die Botschaft nicht aus der Systemwarteschlange entfernt.

wParam: Bestimmt einen *NULL*-Wert.

lParam: Zeigt auf eine *HARDWAREHOOKSTRUCT*-Struktur. Die *HARDWAREHOOKSTRUCT*-Struktur hat folgende Form:

```
typedef struct tagHARDWAREHOOKSTRUCT { /* hhs */
      HWND hWnd;
      UINT wMessage;
      WPRAM wParam;
      LPARAM lParam;
} HARDWAREHOOKSTRUCT;
```

Rückgabewert

Die Callback-Funktion sollte Null zurückgeben, um dem System die Bearbeitung der Botschaft zu erlauben. Der Rückgabewert sollte 1 sein, wenn die Botschaft verworfen werden soll.

Hinweis

Diese Callback-Funktion sollte keinen Wiedergabefilter installieren, da die Funktion nicht *GetMessageExtraInfo* benutzen kann, um die einer Botschaft zugeordneten zusätzlichen Informationen zu ermitteln.

Die Callback-Funktion muß die Pascal-Aufrufkonventionen verwenden und als **far** deklariert sein. Anwendungen müssen die Callback-Funktion installieren, indem sie den Filtertyp *WH_HARDWARE* und die Adresse der Prozedurinstanz der Callback-Funktion in einem Aufruf der Funktion *SetWindowsHookEx* angeben.

HardwareProc ist ein Platzhalter für den bibliotheksdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei (.DEF) der Bibliothek exportiert werden.

Siehe auch

CallNextHookEx, GetMessageExtraInfo, SetWindowsHookEx

hardware_event 3.1

Syntax

```
extrn hardware_event :far

mov   ax, Msg             ; Botschaft
mov   cx, ParamL          ; Niederwertiges WORD von
                          ; lParam der Botschaft
mov   dx, ParamH          ; Höherwertiges WORD von
                          ; lParam der Botschaft
mov   si, hwnd            ; Handle des Zielfensters
mov   di, wParam          ; wParam der Botschaft
cCall hardware_event
```

Beschreibung

Die Funktion *hardware_event* stellt eine hardware-bezogene Botschaft in die Botschaftswarteschlange des Systems. Sie ermöglicht es einem Gerätetreiber für nicht standardisierte Hardware, eine Botschaft in die Warteschlange zu schreiben.

Parameter

Msg: Bestimmt die Botschaft, die in die Botschaftswarteschlange des Systems geschrieben werden soll.

ParamL: Bestimmt das niederwertige *WORD* des Parameters *lParam* der Botschaft.

ParamH: Bestimmt das höherwertige *WORD* des Parameters *lParam* der Botschaft.

hwnd: Bezeichnet das Fenster, dem die Botschaft übergeben wird. Dieser Wert wird außerdem zum niederwertigen *WORD* des Parameters *dwExtraInfo*, der der Botschaft zugeordnet ist. Anwendungen können den Wert dieses Parameters durch einen Aufruf der Funktion *GetMessageExtraInfo* ermitteln.

wParam: Bestimmt den Parameter *wParam* der Botschaft.

Rückgabewert

Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich ist. Schlägt die Funktion fehl, wird Null zurückgegeben.

Hinweis

Anwendungen sollten diese Funktion nicht dazu aufrufen, Tastatur- oder Mausbotschaften in die Botschaftswarteschlange des Systems zu stellen.

Anwendungen können die Funktion *hardware_event* nur aus Assembler-Routinen heraus aufrufen. Die Funktion muß innerhalb der Anwendung folgendermaßen deklariert werden:

```
extrn hardware_event :far
```

Schließt die Anwendung die Datei CMACROS.INC ein, kann diese Funktion auch so deklariert werden:

```
extrnFP hardware_event.
```

Siehe auch

GetMessageExtraInfo

hmemcpy 3.1

Syntax

```
void hmemcpy(hpvDest, hpvSource, cbCopy)

void _huge* hpvDest;              /* Adresse des
                                     Zielpuffers       */
const void _huge* hpvSource;      /* Adresse des
                                     Quellpuffers      */
long cbCopy;                      /* Anzahl der zu
                                     kopierenden Byte  */
```

Beschreibung Die Funktion *hmemcpy* kopiert Byte aus dem Quellpuffer in einen Zielpuffer. Diese Funktion unterstützt Speicherobjekte vom Typ **huge**. Die Speicherobjekte dürfen also größer als 64 KByte und mit der Funktion *GlobalAlloc* reserviert worden sein.

Parameter *hpvDest*: Zeigt auf einen Puffer, der die kopierten Byte aufnimmt.

hpvSource: Zeigt auf einen Puffer, der die zu kopierenden Byte enthält.

cbCopy: Bestimmt die Anzahl der zu kopierenden Byte.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Siehe auch *_hread, _hwrite, lstrcpy*

_hread 3.1

Syntax

```
long _hread(hf, hpvBuffer, cbBuffer)

HFILE hf;                 /* Datei-Handle              */
void _huge* hpvBuffer;    /* Adresse des Puffers für
                             die eingelesenen Daten    */
long cbBuffer;            /* Länge des Datenpuffers    */
```

Beschreibung Die Funktion *_hread* liest Daten aus der angegebenen Datei. Diese Funktion unterstützt Speicherobjekte vom Typ **huge**. Dies sind Objekte, die größer als 64 KByte sind und mit der Funktion *GlobalAlloc* reserviert wurden.

Parameter *hf:* Bezeichnet die zu lesende Datei.

hpvBuffer: Zeigt auf einen Puffer, der die aus der Datei gelesenen Daten aufnehmen soll.

cbBuffer: Bestimmt die Anzahl der Byte, die aus der Datei gelesen werden sollen.

Rückgabewert Der Rückgabewert ist die Anzahl der Byte, die die Funktion aus der Datei gelesen hat. Ist die Anzahl der gelesenen Byte kleiner als der in *cbBuffer* angegebene Wert, hat die Funktion das Ende der Datei (*EOF*) erreicht, bevor die angegebene Anzahl von

Byte gelesen werden konnte. Der Rückgabewert ist *HFILE_ERROR*, wenn die Funktion fehlschlägt.

Siehe auch *_lread, hmemcpy, _hwrite*

_hwrite 3.1

Syntax

```
long _hwrite(hf, hpvBuffer, cbBuffer)

HFILE hf;                       /* Datei-Handle        */
const void _huge* hpvBuffer;    /* Adresse des Puffers
                                   für die zu
                                   schreibenden Daten  */
long cbBuffer;                  /* Größe der Daten     */
```

Beschreibung Die Funktion *_hwrite* schreibt Daten in die angegebene Datei. Diese Funktion unterstützt Speicherobjekte vom Typ **huge**. Das sind Objekte, die größer als 64 KByte sind und mit der Funktion *GlobalAlloc* reserviert wurden.

Parameter *hf:* Bezeichnet die Datei, in die geschrieben werden soll.

hpvBuffer: Zeigt auf einen Puffer, der die in die Datei zu schreibenden Daten enthält.

cbBuffer: Bestimmt die Anzahl der Byte, die in die Datei geschrieben werden sollen.

Rückgabewert Der Rückgabewert ist die Anzahl die Byte, die in die Datei geschrieben wurde. Tritt ein Fehler auf, ist der Rückgabewert *HFILE_ERROR*.

Hinweis Der durch *hpvBuffer* angegebene Puffer darf Segmentgrenzen nicht überschreiten.

Siehe auch *hmemcpy, _hread, _lwrite*

InterruptRegister 3.1

Syntax

```
#include <toolhelp.h>
BOOL InterruptRegister(htask, lpfnIntCallback)

HTASK htask;              /* Handle der Task          */
FARPROC lpfnIntCallback;  /* Adresse der Callback-
                             Funktion                 */
```

Beschreibung Die Funktion *InterruptRegister* installiert eine Callback-Funktion, um alle System-Interrupts zu bearbeiten.

Parameter *htask:* Bezeichnet die Task, die die Callback-Funktion registriert. Der Wert von *htask* dient der Registrierung, nicht der Filterung von Interrupts. Normalerweise ist dieser Wert *NULL* und verweist so auf die aktuelle Task. Der Wert ist nur dann

ungleich *NULL*, wenn eine Anwendung mehr als einen Interrupt-Handler benötigt.

lpfnIntCallback: Zeigt auf die Interrupt-Callback-Funktion, die die Interrupts bearbeitet. Diese Funktion wird immer dann von der ToolHelp-Bibliothek aufgerufen, wenn eine Task einen Interrupt empfängt. Der Wert von *lpfnIntCallback* ist normalerweise der Rückgabewert eines Aufrufs der Funktion *MakeProcInstance*. Dadurch enthält beim Aufruf der Interrupt-Callback-Funktion das Register AX den Selektor des Datensegments der Anwendung. Normalerweise enthält der Prolog einer exportierten Funktion folgenden Code:

```
mov   ds,ax
```

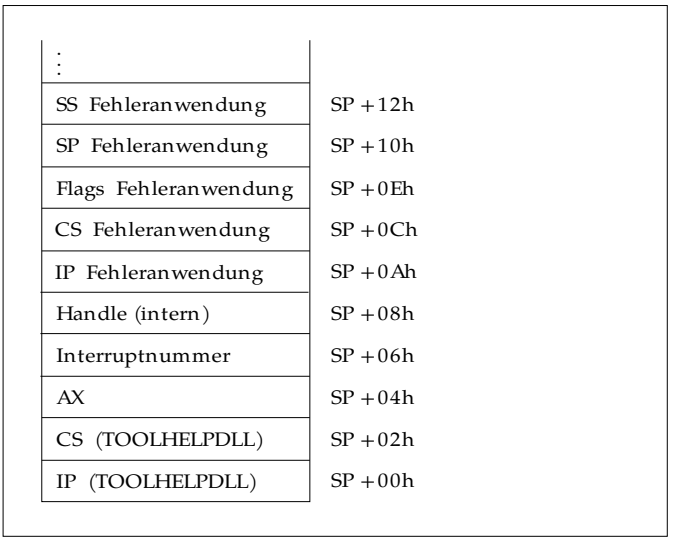
Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Syntax der durch *lpfnIntCallback* bezeichneten Funktion sieht folgendermaßen aus:

```
void InterruptRegisterCallback(void)
```

InterruptRegisterCallback ist ein Platzhalter für den anwendungsdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Anwendung exportiert werden.

Callback-Funktionen zur Bearbeitung von Interrupts müssen wiedereintretend sein und mit einer Seitensperre versehen werden. Außerdem müssen sie explizit alle Registerwerte sichern. Wenn die Funktion von der ToolHelp-Bibliothek aufgerufen wird, sieht der Stack folgendermaßen aus:



Die Werte von SS und SP befinden sich nicht auf dem Stack, solange kein Low-Stack-Fehler auftritt. Dieser Fehler wird dadurch angezeigt, daß das oberste Bit der Interrupt-Nummer gesetzt ist.

Wenn Windows eine Callback-Funktion aufruft, enthält das Register AX den DS-Wert für die Instanz der Anwendung, die die Callback-Funktion enthält. Weitere Informationen über diesen Prozeß finden Sie in der Beschreibung der Funktion *MakeProcInstance*.

Normalerweise wird eine Interrupt-Callback-Funktion exportiert. Wird sie nicht exportiert, sollte der Programmierer sicherstellen, daß der entsprechende Stack-Frame einschließlich des richtigen DS-Werts erzeugt wird.

Eine Interrupt-Callback-Funktion muß alle Registerwerte sichern und anschließend wiederherstellen. Außerdem muß die Funktion eine der folgenden Aufgaben durchführen:

- eine *retf*-Anweisung ausführen, wenn sie den Interrupt nicht bearbeitet . Die Bibliothek TOOLHELP.DLL übergibt den Interrupt an den nächsten Handler in der Liste der Interrupt-Handler;

- die Anwendung mit Hilfe der Funktion *TerminateApp* beenden;

- das Problem beseitigen, das den Interrupt ausgelöst hat, die ersten 10 Byte des Stack löschen und eine *iret*-Anweisung ausführen. Diese Operation nimmt die Ausführung an der angegebenen Adresse wieder auf. Anwendungen können diese Adresse ändern, sollte dies nötig werden;

- ein nichtlokales Goto an eine bekannte Position in der Anwendung ausführen, indem die Funktionen *Catch* und *Throw* benutzt werden. Diese Art von Interrupt-Bearbeitung kann gefährlich sein. Das System kann sich anschließend in einem unstabilen Zustand befinden, weitere Fehler können auftreten. Anwendungen, die Interrupts in dieser Art bearbeiten, müssen sicherstellen, daß der Fehler vom Code der Anwendung verursacht wurde.

Die Bibliothek TOOLHELP.DLL unterstützt die folgenden Interrupts:

Name	Nummer	Bedeutung
INT_DIV0	0	Divsion durch Null
INT_1	1	Debugger-Interrupt
INT_3	3	Breakpoint-Interrupt
INT_UDINSTR	6	Ungültiger Opcode

Name	Nummer	Bedeutung
INT_STKFAULT	12	Stack-Fehler
INT_GPFAULT	13	Allgemeine Verletzung des Speicherschutzes
INT_BADPAGE-FAULT	14	Seitenfehler, der nicht durch einen normalen Zugriff auf den virtuellen Speicher verursacht wurde.
INT_CTLALTSYSRQ	256	Benutzer hat Strg Alt Systemabfrage gedrückt.

Die Bibliothek TOOLHELP.DLL gibt Interrupt-Nummern als *WORD*-Werte zurück. Normale Software-Interrupts und Prozessorfehler werden durch Zahlen im Bereich zwischen 0 und 255 repräsentiert. Die Interrupts der Bibliothek erhalten höhere Werte.

Wenn Sie den Interrupt 256 (Strg Alt Systemabfrage) implementieren wollen, um in den Debugger zu verzweigen, müssen Sie sehr sorgfältig vorgehen. Der Punkt des Abbruchs kann durchaus in einem empfindlichen Teil des Windows-Kernel liegen. Alle *InterruptRegister*-Callback-Funktionen müssen seitengesperrt sein, um Probleme bei der Verwendung dieses Interrupt zu vermeiden. Außerdem kann der Debugger möglicherweise keine Anwenderschnittstellenfunktionen bereitstellen. In jedem Fall kann der Debugger aber Funktionen der ToolHelp-Bibliothek benutzen, um Breakpoints zu setzen und Informationen zu sammeln. Möglicherweise kann der Debugger auch ein Debug-Terminal oder einen zweiten Bildschirm ansprechen, um Informationen darzustellen.

Low-Stack-Fehler

Ein Low-Stack-Fehler tritt auf, wenn auf dem Stack der den Fehler verursachenden Anwendung nicht genügend Speicherplatz zur Verfügung steht. Ein solcher Fehler tritt z.B. dann auf, wenn weniger als 128 Byte auf dem Stack verfügbar sind oder wenn eine fehlerhafte oder zu tiefe Rekursion den Stack aufgebraucht hat.

Ein Low-Stack-Fehler wird dadurch angezeigt, daß das höchste Bit der Interrupt-Nummer gesetzt ist. Tritt z.B. ein Stack-Fehler auf und wird dadurch der SP-Wert ungültig, gibt die ToolHelp-Bibliothek als Fehlernummer den Wert 0x800C anstatt 0x000C zurück.

Interrupt-Handler, die Low-Stack-Fehler bearbeiten sollen, müssen berücksichtigen, daß die Bibliothek die Fehlerinformation auf einem anderen Stack übergeben hat als dem, der zur fehlerauslösenden Anwendung gehört. Der Wert SS:SP befin-

det sich auf dem Stack, weil er vor den übrigen Informationen auf den Stack gelegt wurde. Dieser Wert dient nur Informationszwecken.

Ein Interrupt-Handler sollte bei einem Low-Stack-Fehler die den Fehler verursachende Anweisung niemals erneut ausführen. Dies würde zu einem Absturz des Systems führen. Der Handler kann die Anwendung mit *TerminateApp* beenden oder den Fehler an den nächsten Handler in der Liste der Interrupt-Handler übergeben.

Interrupt-Handler sollten nicht davon ausgehen, daß alle Stack-Fehler Low-Stack-Fehler sind. Greift eine Anwendung z.B. auf eine lokale, auf dem Stack gespeicherte Variable zu, die sich außerhalb ihres Gültigkeitsbereichs befindet, tritt ein Stack-Fehler auf. Dieser Fehlertyp kann wie ein allgemeiner Speicherschutzfehler bearbeitet werden. Ist das höchstwertige Bit der Interrupt-Nummer nicht gesetzt, kann die Anweisung erneut ausgeführt werden.

Interrupt-Handler sollten ebensowenig unterstellen, daß alle Low-Stack-Fehler Stack-Fehler sind. Jeder Fehler, der auftritt, wenn weniger als 128 Byte auf dem Stack verfügbar sind, löst einen Low-Stack-Fehler aus.

Interrupt-Callback-Funktionen, die nicht zur Bearbeitung von Low-Stack-Fehlern entworfen wurden, sollten eine *retf*-Anweisung ausführen. In diesem Fall übergibt die Bibliothek TOOLHELP.DLL den Fehler an den nächstmöglichen Handler in der Liste der Interrupt-Handler.

Siehe auch *Catch, InterruptUnRegister, NotifyRegister, NotifyUnRegister, TerminateApp, Throw*

InterruptUnRegister 3.1

Syntax

```
#include <toolhelp.h>
BOOL InterruptUnRegister(htask)

HTASK htask;          /* Handle der Task              */
```

Beschreibung Die Funktion *InterruptUnRegister* stellt die voreingestellte Bearbeitung von System-Interrupts wieder her.

Parameter *htask:* Bezeichnet die Task. Ist dieser Wert *NULL*, bezeichnet er die aktuelle Task.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Nach der Bearbeitung dieser Funktion übergibt die Bibliothek TOOLHELP.DLL alle Interrupts an den Interrupt-Handler des Systems.

Siehe auch *InterruptRegister, NotifyRegister, NotifyUnRegister, TerminateApp*

IsBadCodePtr 3.1

Syntax
```
BOOL IsBadCodePtr(lpfn)

FARPROC lpfn;          /* zu testender Zeiger          */
```

Beschreibung Die Funktion *IsBadCodePtr* ermittelt, ob ein Zeiger auf ausführbaren Code gültig ist.

Parameter *lpfn:* Zeigt auf eine Funktion.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn der Zeiger nicht auf ausführbaren Code zeigt. Der Rückgabewert ist Null, wenn der Zeiger auf ausführbaren Code zeigt.

Siehe auch *IsBadHugeReadPtr, IsBadHugeWritePtr, IsBadReadPtr, IsBadStringPtr, IsBadWritePtr*

IsBadHugeReadPtr 3.1

Syntax
```
BOOL IsBadHugeReadPtr(lp, cb)

const void _huge* lp; /* zu testender Zeiger         */
DWORD cb;             /* Anzahl der reservierten Byte */
```

Beschreibung Die Funktion *IsBadHugeReadPtr* ermittelt, ob ein **huge**-Zeiger auf lesbaren Speicher gültig ist.

Parameter *lp:* Zeigt auf den Anfang eines Blocks reservierten Speichers. Das Datenobjekt kann sich an beliebiger Position im Speicher befinden und größer als 64 KByte sein.

cb: Die Anzahl der Byte im Speicher, die reserviert wurden.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn der Zeiger nicht auf lesbaren Speicher der angegebenen Größe zeigt. Andernfalls ist der Rückgabewert Null.

Siehe auch *IsBadCodePtr, IsBadHugeWritePtr, IsBadReadPtr, IsBadStringPtr, IsBadWritePtr*

IsBadHugeWritePtr 3.1

Syntax
```
BOOL IsBadHugeWritePtr(lp, cb)

void _huge* lp;       /* zu testender Zeiger          */
DWORD cb;             /* Anzahl der reservierten Byte */
```

Beschreibung Die Funktion *IsBadHugeWritePtr* ermittelt, ob ein **huge**-Zeiger auf beschreibbaren Speicher gültig ist.

Parameter *lp:* Zeigt auf den Anfang eines Blocks reservierten Speichers. Das Datenobjekt kann sich an einer beliebigen Speicherposition befinden und darf größer als 64 KByte sein.

cb: Bestimmt die Anzahl der Byte im Speicher, die reserviert wurden.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn der Zeiger nicht auf beschreibbaren Speicher der angegebenen Größe zeigt. Andernfalls ist der Rückgabewert Null.

Siehe auch *IsBadCodePtr, IsBadHugeReadPtr, IsBadReadPtr, IsBadStringPtr, IsBadWritePtr*

IsBadReadPtr 3.1

Syntax

```
BOOL IsBadReadPtr(lp, cb)

const void FAR* lp;   /* zu testender Zeiger          */
UINT cb;              /* Anzahl der reservierten Byte */
```

Beschreibung Die Funktion *IsBadReadPtr* ermittelt, ob ein Zeiger auf lesbaren Speicher gültig ist.

Parameter *lp:* Zeigt auf den Anfang eines reservierten Speicherblocks.

cb: Bestimmt die Anzahl der Byte im Speicher, die reserviert wurden.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn der Zeiger nicht auf lesbaren Speicher der angegebenen Größe zeigt. Andernfalls ist der Rückgabewert Null.

Siehe auch *IsBadCodePtr, IsBadHugeReadPtr, IsBadHugeWritePtr, IsBadStringPtr, IsBadWritePtr*

IsBadStringPtr 3.1

Syntax

```
BOOL IsBadStringPtr(lpsz, cchMax)

const void FAR* lpsz; /* zu testender Zeiger          */
UINT cchMax;          /* maximale Größe des String    */
```

Beschreibung Die Funktion *IsBadStringPtr* ermittelt, ob ein Zeiger auf einen String gültig ist.

Parameter *lpsz:* Zeigt auf einen nullterminierten String.

cchMax: Bestimmt die maximale Anzahl der Byte im String.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn der Zeiger nicht auf einen String der angegebenen Größe zeigt. Andernfalls ist der Rückgabewert Null.

Siehe auch *IsBadCodePtr, IsBadHugeReadPtr, IsBadHugeWritePtr, IsBadReadPtr, IsBadWritePtr*

IsBadWritePtr 3.1

Syntax
```
BOOL IsBadWritePtr(lp, cb)

void FAR* lp;          /* zu testender Zeiger          */
UINT cb;               /* Anzahl der reservierten Byte */
```

Beschreibung Die Funktion *IsBadWritePtr* ermittelt, ob ein Zeiger auf beschreibbaren Speicher gültig ist.

Parameter *lp:* Zeigt auf den Anfang eines reservierten Speicherblocks.

cb: Die Anzahl der Byte im Speicher, die reserviert wurden.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn der Zeiger nicht auf einen beschreibbaren Speicherblock der angegebenen Größe zeigt. Andernfalls ist der Rückgabewert Null.

Siehe auch *IsBadCodePtr, IsBadHugeReadPtr, IsBadHugeWritePtr, IsBadReadPtr, IsBadStringPtr*

IsGDIObject 3.1

Syntax
```
BOOL IsGDIObject(hobj)

HGDIOBJ hobj;          /* Handle eines Objekts         */
```

Beschreibung Die Funktion *IsGDIObject* ermittelt, ob das angegebene Handle kein Handle eines GDI-Objekts ist.

Parameter *hobj:* Bestimmt das zu überprüfende Handle.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn es sich bei dem Handle um das Handle eines GDI-Objekts handeln könnte. Er ist Null, wenn das Handle nicht das Handle eines GDI-Objekts ist.

Hinweis Die Funktion *IsGDIObject* garantiert nicht, daß es sich beim angegebenen Handle um ein GDI-Objekt handelt. Sie kann nur sicherstellen, daß ein angegebenes Handle nicht das eines GDI-Objekts ist.

Siehe auch *GetObject*

IsMenu 3.1

Syntax
```
BOOL IsMenu(hmenu)

HMENU hmenu;           /* Handle des Menüs             */
```

Beschreibung Die Funktion *IsMenu* ermittelt, ob das angegebene Handle ein Menü-Handle ist.

Parameter *hmenu:* Bezeichnet das zu überprüfende Handle.

Rückgabewert Der Rückgabewert ist Null, wenn das Handle sicher kein Menü-Handle ist. Ein Rückgabewert ungleich Null stellt nicht sicher, daß das Handle ein Menü-Handle ist. In diesem Fall sollte die Anwendung weitere Überprüfungen durchführen.

Hinweis Anwendungen sollten diese Funktion nur benutzen, um sicherzustellen, daß ein angegebenes Handle kein Menü-Handle ist.

Siehe auch *CreateMenu, CreatePopupMenu, DestroyMenu, GetMenu*

IsTask 3.1

Syntax

```
BOOL IsTask(htask)

HTASK htask;        /* Handle der Task                */
```

Beschreibung Die Funktion *IsTask* überprüft die Gültigkeit eines Task-Handle.

Parameter *htask:* Bezeichnet eine Task.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn das Task-Handle gültig und Null, wenn es ungültig ist.

JournalPlaybackProc 3.1

Syntax

```
LRESULT CALLBACK JournalPlaybackProc(code, wParam,
   lParam)

int code;            /* Botschaftsbearbeitungs-Flag  */
WPARAM wParam;       /* nicht definiert              */
LPARAM lParam;       /* Adresse der Struktur für
                        die Botschaft                */
```

Beschreibung Die Funktion *JournalPlaybackProc* ist eine bibliotheksdefinierte Callback-Funktion, die von Bibliotheken benutzt werden kann, um Maus- und Tastaturbotschaften in die Botschaftswarteschlange des Systems zu schreiben. Normalerweise verwendet eine Bibliothek diese Funktion, um Maus- und Tastaturbotschaften abzuspielen, die vorher mit der Funktion *JournalRecordProc* aufgezeichnet wurden. Solange die Funktion *JournalPlaybackProc* installiert ist, sind normale Maus- und Tastatureingaben deaktiviert.

Parameter *code:* Bestimmt, ob die Callback-Funktion die Botschaft bearbeiten oder die Funktion *CallNextHookEx* aufrufen soll. Ist dieser Parameter kleiner als Null, sollte die Callback-Funktion die Botschaft ohne weitere Bearbeitung an *CallNextHookEx* übergeben.

wParam: Bestimmt einen *NULL*-Wert.

lParam: Zeigt auf eine *EVENTMSG*-Struktur, die die von der Callback-Funktion zu bearbeitende Botschaft repräsentiert. Die *EVENTMSG*-Struktur hat folgende Form:

```
typedef struct tagEVENTMSG { /* em */
      UINT message;
      UINT paramL;
      UINT paramH;
      DWORD time;
} EVENTMSG;
```

Rückgabewert Die Callback-Funktion sollte einen Wert zurückgeben, der die Zeit in Takteinheiten repräsentiert, die das System mit der Bearbeitung der Botschaft warten sollte. Dieser Wert kann durch Ermittlung der Differenz zwischen den Elementen *time* der aktuellen und der vorhergehenden Eingabebotschaft berechnet werden. Gibt die Funktion Null zurück, wird die Botschaft sofort bearbeitet.

Hinweis Die Funktion *JournalPlaybackProc* sollte Eingabebotschaften in den Parameter *lParam* kopieren. Die Botschaft muß mit der Funktion *JournalRecordProc* aufgezeichnet, darf von dieser aber nicht bearbeitet werden.

Nachdem die Funktion die Steuerung an das System zurückgegeben hat, werden die Botschaften wieder bearbeitet. Hat der Parameter *code* den Wert *HC_SKIP*, sollte die Filterfunktion die folgende aufgezeichnete Ereignisbotschaft bei ihrem nächsten Aufruf zurückgeben.

Diese Callback-Funktion sollte sich in einer dynamischen Linkbibliothek befinden.

Anwendungen müssen die Callback-Funktion installieren, indem sie den Filtertyp *WH_JOURNALPLAYBACK* und die Adresse der Prozedurinstanz der Callback-Funktion in einem Aufruf der Funktion *SetWindowsHookEx* übergeben.

JournalPlaybackProc ist ein Platzhalter für den bibliotheksdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Bibliothek exportiert werden.

Siehe auch *CallNextHookEx, JournalRecordProc, SetWindowsHookEx*

JournalRecordProc 3.1

Syntax

```
LRESULT CALLBACK JournalRecordProc(code, wParam, lParam)

int code;            /* Botschaftsbearbeitungs-Flag  */
WPARAM wParam;       /* nicht definiert              */
LPARAM lParam;       /* Adresse der Struktur für
                        die Botschaft                */
```

Beschreibung Die Funktion *JournalRecordProc* ist eine bibliotheksdefinierte Callback-Funktion zum Aufzeichnen von Botschaften, die vom System aus der Botschaftwarteschlange des Systems entfernt werden. Bibliotheken können später die Funktion *JournalPlaybackProc* aufrufen, um die Botschaften abzuspielen.

Parameter *code:* Bestimmt, ob die Callback-Funktion die Botschaft bearbeiten oder die Funktion *CallNextHookEx* aufrufen soll. Ist dieser Parameter kleiner als Null, sollte die Callback-Funktion die Botschaft ohne weitere Bearbeitung an *CallNextHookEx* übergeben.

wParam: Bestimmt einen *NULL*-Wert.

lParam: Zeigt auf eine *MSG*-Struktur. Die *MSG*-Struktur hat folgende Form:

```
typedef struct tagMSG { /* msg */
    HWND hwnd;
    UINT message;
    WPARAM wParam;
    LPARAM lParam;
    DWORD time;
    POINT pt;
} MSG;
```

Rückgabewert Die Callback-Funktion sollte Null zurückgeben.

Hinweis *JournalRecordProc*-Callback-Funktionen sollten die Botschaften nur kopieren, nicht verändern. Nachdem die Steuerung an das System zurückgegeben wurde, werden wieder Botschaften bearbeitet. Die Callback-Funktion benötigt keinen Rückgabewert. Diese Callback-Funktion muß sich in einer dynamischen Linkbibliothek befinden.

Anwendungen müssen die Callback-Funktion installieren, indem sie den Filtertyp *WH_JOURNALRECORD* und die Adresse der Prozedurinstanz der Callback-Funktion in einem Aufruf der Funktion *SetWindowsHookEx* übergeben.

JournalRecordProc ist ein Platzhalter für den bibliotheksdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Bibliothek exportiert werden.

Siehe auch *CallNextHookEx, JournalPlaybackProc, SetWindowsHookEx*

KeyboardProc 3.1

Syntax

```
LRESULT CALLBACK KeyboardProc(code, wParam, lParam)

int code;          /* Botschaftsbearbeitungs-Flag     */
WPARAM wParam      /* virtueller Tastencode           */
LPARAM lParam      /* Tastaturbotschaftsinformationen */
```

Beschreibung

Die Funktion *KeyboardProc* ist eine bibliotheksdefinierte Callback-Funktion, die vom System immer dann aufgerufen wird, wenn die Anwendung die Funktion *GetMessage* oder *PeekMessage* aufruft und gleichzeitig eine *WM_KEYUP*- oder *WM_KEYDOWN*-Tastaturbotschaft zu bearbeiten ist.

Parameter

code: Bestimmt, ob die Callback-Funktion die Botschaft bearbeiten oder die Funktion *CallNextHookEx* aufrufen soll. Ist dieser Wert *HC_NOREMOVE*, hat die Anwendung die Funktion *PeekMessage* mit der Option *PM_NOREMOVE* aufgerufen. In diesem Fall wird die Botschaft nicht aus der Warteschlange des Systems entfernt. Ist der Wert kleiner als Null, sollte die Callback-Funktion die Botschaft ohne weitere Bearbeitung an *CallNextHookEx* übergeben.

wParam: Bestimmt den virtuellen Tastencode der angegebenen Taste.

lParam: Bestimmt den Wiederholungszähler, den Scan-Code, die erweiterten Tasten, den vorhergehenden Tastenstatus und den Kontextcode. Bit 0 ist das niederwertige Bit:

Bit	Beschreibung
0-15	Bestimmt den Wiederholungszähler. Dieser Wert ist die Anzahl der Tastenwiederholungen, die aus der Dauer des Tastendrucks ermittelt wird.
16-23	Bestimmt den Scan-Code. Dieser Wert ist herstellerspezifisch (OEM).
24	Bestimmt, ob es sich bei der Taste um eine erweiterte Taste handelt, wie z.B. eine Funktionstaste oder eine Taste im numerischen Block. Dieser Wert ist 1, wenn es sich um eine erweiterte Taste handelt, andernfalls 0.
25-26	Nicht verwendet.
27-28	Werden intern von Windows verwendet.
29	Bestimmt den Kontextcode. Der Wert ist 1, wenn die Taste Alt gehalten wird, während die Taste gedrückt wird. Andernfalls ist dieser Wert 0.
30	Bestimmt den vorhergehenden Tastenstatus. Der Wert ist 1, wenn die Taste gedrückt ist, bevor die Botschaft übergeben wird. Er ist 0, wenn die Taste nicht gedrückt ist.
31	Dieser Wert ist 1, wenn die Taste losgelassen oder 0, wenn die Taste gedrückt wird.

Rückgabewert

Die Callback-Funktion sollte 0 zurückgeben, wenn die Botschaft vom System bearbeitet werden soll. Der Rückgabewert sollte 1 sein, wenn die Botschaft verworfen werden soll.

Hinweis

Diese Callback-Funktion muß sich in einer dynamischen Linkbibliothek befinden.

Anwendungen müssen die Callback-Funktion installieren, indem sie den Filtertyp *WH_KEYBOARD* und die Adresse der Prozedurinstanz der Callback-Funktion in einem Aufruf der Funktion *SetWindowsHookEx* angeben.

KeyboardProc ist ein Platzhalter für den bibliotheksdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Bibliothek exportiert werden.

Siehe auch

CallNextHookEx, GetMessage, PeekMessage, SetWindowsHookEx

LibMain 2.x

Syntax

```
int CALLBACK LibMain(hinst, wDataSeg, cbHeapSize,
   lpszCmdLine)

HINSTANCE hinst;    /* Handle der Bibliotheksinstanz  */
WORD wDataSeg;      /* Datensegment der Bibliothek    */
WORD cbHeapSize;    /* voreingestellte Größe des Heap */
LPSTR lpszCmdLine;  /* Kommandozeilenoptionen         */
```

Beschreibung

Die Funktion *LibMain* wird vom System aufgerufen, um eine dynamische Linkbibliothek (DLL) zu initialisieren. Eine DLL muß die Funktion *LibMain* enthalten, wenn sie die Datei LIBENTRY.OBJ einbindet.

Parameter

hinst: Bezeichnet die Instanz der DLL.

wDataSeg: Bestimmt den Wert des Datensegment-Registers (DS).

cbHeapSize: Bestimmt die Größe des Heap, die in der Moduldefinitionsdatei definiert ist. Die Routine *LibEntry* in der Datei LIBENTRY.OBJ verwendet diesen Wert, um den lokalen Heap zu initialisieren.

lpszCmdLine: Zeigt auf einen nullterminierten String, der Kommandozeilenoptionen enthält. Dieser Parameter wird nur selten von dynamischen Linkbibliotheken benutzt.

Rückgabewert

Ist die Funktion erfolgreich, sollte sie 1 zurückgeben, andernfalls 0.

Hinweis

Die Funktion *LibMain* wird von *LibEntry*, diese wiederum von Windows aufgerufen, wenn die DLL geladen wird. Die Routine *LibEntry* wird vom Modul LIBENTRY.OBJ bereitgestellt. *LibEntry* initialisiert den Heap der DLL – wenn ein *HEAPSIZE*-Wert in der Moduldefinitionsdatei der DLL angegeben ist – bevor die Funktion *LibMain* aufgerufen wird.

Beispiel

Das folgende Beispiel zeigt eine typische *LibMain*-Funktion.

```
int CALLBACK LibMain(HINSTANCE hinst, WORD wDataSeg,
    WORD cbHeap, LPSTR lpszCmdLine)

{
   HGLOBAL            hgblClassStruct;
   LPWNDCLASS         lpClassStruct;
   static HINSTANCE   hinstLib;

   /* Wurde die Bibliothek bereits initialisiert? */

   if (hinstLib == NULL) {
     hgblClassStruct = GlobalAlloc (GHND,
                        sizeof(WNDCLASS));
     if (hgblClassStruct != NULL) {
       lpClassStruct = (LPWNDClASS)
                       GlobalLock(hgblClassStruct);
       if (lpClassStruct != NULL) {

          /* Attribute der Klasse definieren. */

         lpClassStruct->style = CS_HREDRAW |
           CS_VREDRAW | CS_DBLCLKS | GS_GLOBALCLASS;
         lpClassStruct->lpfnWndProc = DllWndProc;
         lpClassStruct->cbWndExtra = 0;
         lpClassStruct->hInstance = hinst;
         lpClassStruct->hIcon = NULL;
         lpClassStruct->hCursor = LoadCursor(NULL,
                                     IDC_ARROW);
         lpClassStruct->hbrBackground = (HBRUSH)
                        (COLOR_WINDOW + 1);
         lpClassStruct->lpszMenuName = NULL;
         lpClassStruct->lpszClassName =  "MyClassName";

         hinstLib = (RegisterClass(lpClassStruct)) ?
               hinst: NULL;

         GlobalUnlock(hgblClassStruct);
       }
         GlobalFree(hgblClassStruct);
      }
   }
   return (hinstLib ? 1 : 0);

          /* return 1 = erfolgreiche Ausführung;
                    0 = fehlgeschlagen */

}
```

Siehe auch

GlobalAlloc, GlobalFree, GlobalLock, GlobalUnlock, WEP

LineDDAProc 3.1

Syntax

```
void CALLBACK LineDDAProc(xPos, yPos, lpData)

int xPos;            /* x-Koordinate der aktuellen
                        Position                    */
int yPost;           /* y-Koordinate der aktuellen
                        Position                    */
LPARAM lpData;       /* Adresse der anwendungs-
                        definierten Daten           */
```

Beschreibung *LineDDAProc* ist eine anwendungsdefinierte Callback-Funktion, die Koordinaten der Funktion *LineDDA* verarbeitet.

Parameter *xPos:* Bestimmt die x-Koordinate des aktuellen Punkts.

yPos: Bestimmt die y-Koordinate des aktuellen Punkts.

lpData: Zeigt auf die anwendungsdefinierten Daten.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Hinweis Anwendungen müssen diese Funktion registrieren, indem sie ihre Adresse an die Funktion *LineDDA* übergeben.

LineDDAProc ist ein Platzhalter für den anwendungsdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Anwendung exportiert werden.

Siehe auch *LineDDA*

Load Proc 2.x

Syntax

```
HGLOBAL CALLBACK LoadProc(hglbMem, hinst, hrsrcResInfo)

HGLOBAL hglbMem;     /* Handle des Objekts, das
                        die Ressource enthält       */
HINSTANCE hinst;     /* Handle der Anwendungsinstanz */
HRSRC hrsrcResInfo;  /* Handle einer Ressource      */
```

Beschreibung Die Funktion *LoadProc* ist eine anwendungsdefinierte Callback-Funktion, die Informationen darüber erhält, ob eine Ressource gesperrt werden soll. Sie kann diese Informationen wie gewünscht bearbeiten.

Parameter *hglbMem:* Bezeichnet ein Speicherobjekt, das eine Ressource enthält. Dieser Parameter ist *NULL*, wenn die Ressource noch nicht geladen wurde.

hinst: Bezeichnet die Instanz des Moduls, dessen ausführbare Datei die Ressource enthält.

hrsrcResInfo: Bezeichnet die Ressource. Die Ressource muß mit der Funktion *FindResource* erzeugt worden sein.

Rückgabewert Der Rückgabewert ist ein globales Speicher-Handle auf Speicher, der unter Verwendung des Flag *GMEM_DDESHARE* mit der Funktion *GlobalAlloc* reserviert wurde.

Hinweis Schlägt der Versuch fehl, das durch den Parameter *hglbMem* bezeichnete Speicherobjekt zu sperren, wurde die Ressource verworfen und muß neu geladen werden.

LoadProc ist ein Platzhalter für den anwendungsdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Anwendung exportiert werden.

Siehe auch *FindResource, GlobalAlloc, SetResourceHandler*

LocalFirst 3.1

Syntax

```
#include <toolhelp.h>
BOOL LocalFirst(lple, hglbHeap)

LOCALENTRY FAR* lple;  /* Adresse einer LOCALENTRY-
                          Struktur                  */
HGLOBAL hglbHeap;      /* Handle des lokalen Heap    */
```

Beschreibung Die Funktion *LocalFirst* füllt die angegebene Struktur mit Informationen, die das erste Objekt auf dem lokalen Heap beschreiben.

Parameter *lple:* Zeigt auf eine *LOCALENTRY*-Struktur, die Informationen über das lokale Speicherobjekt aufnimmt. Die *LOCALENTRY*-Struktur hat folgende Form:

```
#include <toolhelp.h>

typedef struct tagLOCALENTRY { /* le */
    DWORD dwSize;
    HLOCAL hHandle;
    WORD wAddress;
    WORD wSize;
    WORD wFlags;
    WORD wcLock;
    WORD wType;
    WORD hHeap;
    WORD wHeapType;
    WORD wNext;
} LOCALENTRY;
```

hglbHeap: Bezeichnet den lokalen Heap.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *LocalFirst* kann benutzt werden, um eine Überprüfung des lokalen Heap zu beginnen. Anwendungen kön-

nen nachfolgende Objekte auf dem lokalen Heap mit der Funktion *LocalNext* untersuchen.

Anwendungen müssen die *LOCALENTRY*-Struktur initialisieren und ihre Größe im Element *dwSize* angeben, bevor sie *LocalFirst* aufrufen.

Siehe auch *LocalInfo, LocalNext*

LocalInfo 3.1

Syntax

```
#include <toolhelp.h>
BOOL LocalInfo(lpli, hglHeap)

LOCALINFO FAR* lpli;  /* Adresse einer LOCALINFO-
                         Struktur                  */
HGLOBAL hglbHeap;     /* Handle des lokalen Heap   */
```

Beschreibung Die Funktion *LocalInfo* füllt die angegebene Struktur mit Informationen, die den lokalen Heap beschreiben.

Parameter *lpli:* Zeigt auf eine *LOCALINFO*-Struktur, die Informationen über den lokalen Heap aufnimmt. Die *LOCALINFO*-Struktur hat folgende Form:

```
#include <toolhelp.h>

typedef struct tagLOCALINFO { /* li */
    DWORD dwSize;
    WORD wcItems;
} LOCALINFO;
```

hglbHeap: Bezeichnet den lokalen Heap, der beschrieben werden soll.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Informationen in der *LOCALINFO*-Struktur können benutzt werden, um zu ermitteln, wieviel Speicherplatz für eine Überprüfung des lokalen Heap reserviert werden muß.

Anwendungen müssen die *LOCALINFO*-Struktur initialisieren und ihre Größe im Element *dwSize* in Byte angeben, bevor sie *LocalInfo* aufrufen.

Siehe auch *LocalFirst, LocalNext*

LocalNext 3.1

Syntax

```
#include <toolhelp.h>
BOOL LocalNext(lple)

LOCALENTRY FAR* lple;   /* Adresse einer LOCALENTRY-
                           Struktur                  */
```

Beschreibung Die Funktion *LocalNext* füllt die angegebene Struktur mit Informationen, die das nächste Objekt auf dem lokalen Heap beschreiben.

Parameter *lple:* Zeigt auf eine *LOCALENTRY*-Struktur, die Informationen über das lokale Speicherobjekt aufnimmt. Die *LOCALENTRY*-Struktur hat folgende Form:

```
#include <toolhelp.h>

typedef struct tagLOCALENTRY { /* le */
    DWORD dwSize;
    HLOCAL hHandle;
    WORD wAddress;
    WORD wSize;
    WORD wFlags;
    WORD wcLock;
    WORD wType;
    WORD hHeap;
    WORD wHeapType;
    WORD wNext;
} LOCALENTRY;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *LocalNext* kann benutzt werden, um eine Überprüfung des lokalen Heap fortzusetzen, die mit der Funktion *LocalFirst* gestartet wurde.

Siehe auch *LocalFirst, LocalInfo*

LockInput 3.1

Syntax

```
BOOL LockInput(hReserved, hwndInput, fLock)

HANDLE hReserved;     /* reserviert, muß NULL sein    */
HWND hwndInput;       /* Handle des Fensters, das
                         alle Eingaben erhalten soll  */
BOOL fLock;           /* Sperren/Entsperren-Flag      */
```

Beschreibung Die Funktion *LockInput* sperrt Eingaben für alle Tasks außer der aktuellen, wenn der Parameter *fLock* den Wert *TRUE* hat. Das angegebene Fenster wird system-modal. Es erhält alle Eingaben. Hat *fLock* den Wert *FALSE*, entsperrt *LockInput* die Eingaben und stellt den vorhergehenden, nicht gesperrten Status des Systems wieder her.

Parameter *hReserved:* Dieser Parameter ist reserviert und muß *NULL* sein.

hwndInput: Bezeichnet das Fenster, das alle Eingaben erhalten soll. Dieses Fenster muß ein Fenster der aktuellen Task sein. Hat *fLock* den Wert *FALSE*, sollte dieser Parameter *NULL* sein.

fLock: Legt fest, ob die Eingaben gesperrt oder entsperrt werden sollen. Der Wert *TRUE* sperrt die Eingaben, der Wert *FALSE* entsperrt sie.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Windows-Debugger rufen die Funktion *LockInput* mit dem auf *TRUE* gesetzten Parameter *fLock* auf, bevor sie den Hard-Modus aktivieren. Diese Operation sichert den aktuellen globalen Status. Um den Hard-Modus zu verlassen, ruft der Debugger *LockInput* mit dem Wert *FALSE* für *fLock* auf. Dies stellt den globalen Status in dem vor Aktivierung des Hard-Modus bestehenden Zustand wieder her. Debugger müssen den globalen Status vor ihrer Rückkehr wiederherstellen. Aufrufe von *LockInput* können nicht verschachtelt werden.

Siehe auch *DirectedYield*

LockWindowUpdate 3.1

Syntax

```
BOOL LockWindowUpdate(hwndLock)

HWND hwndLock;        /* Handle des Fensters          */
```

Beschreibung Die Funktion *LockWindowUpdate* deaktiviert oder reaktiviert das Zeichnen im angegebenen Fenster. Ein gesperrtes Fenster kann nicht verschoben werden. Es kann nur ein Fenster zur Zeit gesperrt werden.

Parameter *hwndLock:* Bezeichnet das Fenster, in dem das Zeichnen deaktiviert wird. Wenn dieser Parameter *NULL* ist, wird das Zeichnen im gesperrten Fenster wieder möglich.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null. Er ist Null, wenn ein Fehler auftritt oder die Funktion *LockWindowUpdate* bereits zum Sperren eines anderen Fensters aufgerufen wurde.

Hinweis Ruft eine Anwendung mit einem gesperrten Fenster oder einem untergeordneten gesperrten Fenster die Funktion *GetDC*, *GetDCEx* oder *BeginPaint* auf, gibt die aufgerufene Funktion einen Gerätekontext zurück, dessen sichtbarer Bereich leer ist. Dies geschieht solange, bis die Anwendung das Fenster entsperrt, indem sie *LockWindowUpdate* aufruft und dabei *NULL* für *hwndLock* angibt.

Während die Aktualisierung eines Fensters gesperrt ist, überwacht das System das umgebende Rechteck auf Zeichenoperationen im Gerätekontext, der dem gesperrten Fenster zugeordnet ist. Wird das Zeichnen reaktiviert, ist das umgebende Rechteck im gesperrten Fenster und dessen untergeordneten Fenstern ungültig. Dies löst eine *WM_PAINT*-Botschaft

aus, damit der Bildschirm aktualisiert wird. Sind keine Zeichenoperationen durchgeführt worden, während die Aktualisierung des Fensters gesperrt war, wird kein Bereich ungültig.

Die Funktion *LockWindowUpdate* macht das angegebene Fenster nicht unsichtbar und setzt das Stil-Bit *WS_VISIBLE* nicht zurück.

LogError 3.1

Syntax

```
void LogError(uErr, lpvInfo)

UINT uErr;          /* Fehlertyp                    */
void FAR* lpvInfo;  /* Adresse der
                       Fehlerinformationen          */
```

Beschreibung

Die Funktion *LogError* bezeichnet den zuletzt aufgetretenen Systemfehler. *LogError* wird meist von der Interrupt-Callback-Funktion einer Anwendung aufgerufen, um Informationen über den Fehler an den Benutzer zu übergeben.

Parameter

uErr: Bestimmt den Typ des aufgetretenen Fehlers. In Abhängigkeit vom Wert des Parameters *uErr* kann der Parameter *lpvInfo* auf weitere Informationen über den Fehler zeigen. *uErr* kann einen oder mehrere der folgenden Werte annehmen:

Wert	Bedeutung
ERR_ALLOCRES	*AllocResource* ist fehlgeschlagen.
ERR_BADINDEX	An eine der Funktionen *GetClassLong*, *GetClassWord*, *GetWindowLong*, *GetWindowWord*, *SetClassLong*, *SetClassWord*, *SetWindowLong* oder *SetWindowWord* wurde ein falscher Index übergeben.
ERR_BYTE	Ungültiger 8-Bit-Parameter.
ERR_CREATEDC	Die Funktion *CreateCompatibleDC*, *CreateDC* oder *CreateIC* ist fehlgeschlagen.
ERR_CREATEDLG	Dialogfenster konnte nicht erzeugt werden, da *LoadMenu* fehlgeschlagen ist.
ERR_CREATEDLG2	Dialogfenster konnte nicht erzeugt werden, da *CreateWindow* fehlgeschlagen ist.
ERR_CREATEMENU	Menü konnte nicht erzeugt werden.
ERR_CREATEMETA	*CreateMetaFile* ist fehlgeschlagen.
ERR_CREATEWND	Fenster konnte nicht erzeugt werden, da die Klasse nicht gefunden wurde.
ERR_DCBUSY	Cache für Gerätekontexte (DC) ist voll.

Wert	Bedeutung
ERR_DELOBJ-SELECTED	Das Programm versucht ein Bitmap zu löschen, das in einem Gerätekontext ausgewählt ist.
ERR_DWORD	Ungültiger 32-Bit-Parameter.
ERR_GALLOC	*GlobalAlloc* ist fehlgeschlagen.
ERR_GLOCK	*GLobalLock* ist fehlgeschlagen.
ERR_GREALLOC	*GlobalReAlloc* ist fehlgeschlagen.
ERR_LALLOC	*LocalAlloc* ist fehlgeschlagen.
ERR_LLOCK	*LocalLock* ist fehlgeschlagen.
ERR_LOADMENU	*LoadMenu* ist fehlgeschlagen.
ERR_LOADMODULE	*LoadModule* ist fehlgeschlagen.
ERR_LOADSTR	*LoadString* ist fehlgeschlagen.
ERR_LOCKRES	*LockResource* ist fehlgeschlagen.
ERR_LREALLOC	*LocalReAlloc* ist fehlgeschlagen.
ERR_NESTED-BEGINPAINT	Das Programm enthält verschachtelte Aufrufe von *BeginPaint*.
ERR_REGISTER-CLASS	*RegisterClass* ist fehlgeschlagen, da die Klasse bereits registriert ist.
ERR_SELBITMAP	Das Programm versucht ein Bitmap auszuwählen, das bereits ausgewählt ist.
ERR_SIZE_MASK	Bezeichnet die zwei Bits von *uErr*, die die Größe des ungültigen Parameters angeben.
ERR_STRUCEXTRA	Das Programm greift auf nicht reservierten Speicherplatz zu.
ERR_WARNING	Ein nichtschwerwiegender Fehler ist aufgetreten.
ERR_WORD	Ungültiger 16-Bit-Parameter.

lpvInfo: Zeigt auf weitere Informationen über den Fehler. Der Wert von *lpvInfo* hängt vom Wert von *uErr* ab. Wenn der Wert von *uErr* & *ERR_SIZE_MASK* 0 ist, ist *lpvInfo* nicht definiert. Aktuell hat kein möglicher Wert von *uErr* eine definierte Bedeutung für *lpvInfo*.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Hinweis Die von *LogError* bezeichneten Fehler können durch die mit *NotifyRegister* installierte Callback-Funktion abgefangen wer-

den. Fehlerwerte, deren untere 12 Bits kleiner als 0x07FF sind, sind für die Verwendung durch Windows reserviert.

Siehe auch *LogParamError, NotifyRegister*

LogParamError 3.1

Syntax

```
void LogParamError(uErr, lpfn, lpvParam)

UINT uErr;          /* Fehlertyp                    */
FARPROC lpfn;       /* Adresse, an der der Fehler
                       aufgetreten ist              */
void FAR* lpvParam; /* Adresse weiterer
                       Fehlerinformationen          */
```

Beschreibung Die Funktion *LogParamError* bezeichnet den zuletzt aufgetretenen Fehler bei der Überprüfung der Gültigkeit eines Parameters. Normalerweise wird *LogParamError* von der Interrupt-Callback-Funktion einer Anwendung aufgerufen, um Informationen über einen ungültigen Parameter an den Benutzer weiterzugeben.

Parameter *uErr:* Bestimmt den Typ des Fehlers, der bei der Gültigkeitsüberprüfung eines Parameters aufgetreten ist. In Abhängigkeit vom Wert von *uErr* kann der Parameter *lpvParam* auf weitere Informationen über den Fehler zeigen. *uErr* kann einen oder mehrere der folgenden Werte annehmen:

Wert	Bedeutung
ERR_BAD_ATOM	Ungültiges Atom.
ERR_BAD_CID	Ungültiger Bezeichner für serielle Schnittstelle (CID).
ERR_BAD_COORDS	Ungültige x-, y-Koordinaten.
ERR_BAD_DFLAGS	Ungültige 32-Bit-Flags.
ERR_BAD_DINDEX	Ungültiger 32-Bit-Index oder Index außerhalb des Gültigkeitsbereichs.
ERR_BAD_DVALUE	Ungültiger vorzeichenloser oder vorzeichenbehafteter 32-Bit-Wert.
ERR_BAD_FLAGS	Ungültige Bit-Flags.
ERR_BAD_FUNC_PT	Ungültiger Funktionszeiger.
ERR_BAD_GDI_OBJECT	Ungültiges GDI-Objekt (GDI = grafische Geräteschnittstelle).
ERR_BAD_GLOBAL_HANDLE	Ungültiges globales Handle.

Wert	Bedeutung
ERR_BAD_HANDLE	Ungültiges generisches Handle.
ERR_BAD_HBITMAP	Ungültiges Bitmap-Handle.
ERR_BAD_HBRUSH	Ungültiges Pinsel-Handle.
ERR_BAD_HCURSOR	Ungültiges Mauszeiger-Handle.
ERR_BAD_HDC	Ungültiges Gerätekontext-Handle (DC).
ERR_BAD_HDRVR	Ungültiges Treiber-Handle.
ERR_BAD_HDWP	Ungültiges Handle einer Fensterpositionsstruktur.
ERR_BAD_HFILE	Ungültiges Datei-Handle.
ERR_BAD_HFONT	Ungültiges Schrift-Handle.
ERR_BAD_HICON	Ungültiges Symbol-Handle.
ERR_BAD_-HINSTANCE	Ungültiges Instanzen-Handle.
ERR_BAD_HMENU	Ungültiges Menü-Handle.
ERR_BAD_-HMETAFILE	Ungültiges Metafile-Handle.
ERR_BAD_-HMODULE	Ungültiges Modul-Handle.
ERR_BAD_H-PALETTE	Ungültiges Paletten-Handle.
ERR_BAD_HPEN	Ungültiges Stift-Handle.
ERR_BAD_HRGN	Ungültiges Region-Handle.
ERR_BAD_HWND	Ungültiges Fenster-Handle.
ERR_BAD_INDEX	Ungültiger Index oder Index außerhalb des Gültigkeitsbereichs.
ERR_BAD_LOCAL_-HANDLE	Ungültiges lokales Handle.
ERR_BAD_PTR	Ungültiger Zeiger.
ERR_BAD_SELEC-TOR	Ungültiger Selektor.
ERR_BAD_STRING_-PTR	Ungültiger Zeiger auf nullterminierten String.
ERR_BAD_VALUE	Ungültiger vorzeichenloser oder vorzeichenbehafteter 16-Bit-Wert.

Wert	Bedeutung
ERR_BYTE	Ungültiger 8-Bit-Parameter.
ERR_DWORD	Ungültiger 32-Bit-Parameter.
ERR_PARAM	Bei der Gültigkeitsüberprüfung eines Parameters ist ein Fehler aufgetreten. Dieses Flag ist immer gesetzt.
ERR_SIZE_MASK	Bezeichnet die beiden Bits von *uErr*, die die Größe des ungültigen Parameters angeben.
ERR_WARNING	Ein ungültiger Parameter wurde ermittelt, der Fehler ist jedoch nicht so schwer, daß die Funktion fehlschlägt. Der ungültige Parameter wird protokolliert, der Aufruf der Funktion läuft jedoch normal ab.
ERR_WORD	Ungültiger 16-Bit-Parameter.

lpfn: Bestimmt die Adresse, an der der Parameterfehler aufgetreten ist. Dieser Wert ist *NULL*, wenn die Adresse unbekannt ist.

lpvParam: Zeigt auf weitere Informationen über den Fehler. Der Wert von *lpvParam* hängt vom Wert von *uErr* ab. Wenn der Wert von *uErr* & *ERR_SIZE_MASK* 0 ist, ist *lpvParam* nicht definiert. Aktuell hat keiner der Werte von *uErr* eine definierte Bedeutung für *lpvParam*.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Hinweis Die von *LogParamError* bezeichneten Fehler können durch die mit *NotifyRegister* installierte Callback-Funktion abgefangen werden.

Fehlerwerte, deren untere 12 Bits kleiner als 0x07FF sind, sind für die Verwendung durch Windows reserviert.

Die Größe des Werts, der in *lpvParam* übergeben wird, ist durch die Werte der durch *ERR_SIZE_MASK* ausgewählten Bits wie folgt bestimmt:

```
switch (err & ERR_SIZE_MASK)
{
case ERR_BYTE:          /* ungültiger 8-Bit-Parameter */
   b = LOBYTE(param);
   break;

case ERR_WORD:          /* ungültiger 16-Bit-Parameter */
   w = LOWORD(param);
   break;

case ERR_DWORD:         /* ungültiger 32-Bit-Parameter */
```

```
      l = (DWORD)param;
      break;

   default:              /* ungültiger Parameterwert
                            ist unbekannt */
      break;
   }
```

LZClose 3.1

Syntax

```
#include <lzexpand.h>
void LZClose(hf)

HFILE hf;        /* Handle der zu schließenden Datei  */
```

Beschreibung Diese Funktion schließt eine Datei, die mit der Funktion *LZOpenFile* oder *OpenFile* geöffnet wurde.

Parameter *hf:* Bezeichnet die Quelldatei.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Hinweis Falls die Datei mit dem Microsoft-Programm COMPRESS.EXE komprimiert und mit der Funktion *LZOpenFile* geöffnet wurde, gibt *LZClose* den Platz auf dem globalen Heap frei, der zum Entkomprimieren der Datei benötigt wurde.

Beispiel Das folgende Beispiel verwendet *LZClose*, um die mit *LZOpenFile* geöffnete Datei zu schließen:

```
char szSrc[] = {"readme.txt"};
char szDst[] = {"readme.bak"};
OFSTRUCT ofStrSrc;
OFSTRUCT ofStrDest;
HFILE hfSrcFile, hfDstFile;

/* ¹ffnet die Quelldatei */

hfSrcFile = LZOpenFile(szSrc, &ofStrSrc, OF_READ);

/* Erzeugt die Zieldatei */

hfDstFile = LZOpenFile (szDst, &ofStrDest, OF_CREATE);

/* Kopiert die Quelldatei zur Zieldatei */

LZCopy(hfSrcFile, hfDstFile);

/* Schließt die Dateien */

LZClose(hfSrcFile);
LZClose(hfDstFile);
```

Siehe auch *OpenFile, LZOpenFile*

LZCopy 3.1

Syntax

```
#include <lzexpand.h>
LONG LZCopy(hfSource, hfDest)

HFILE hfSource      /*  Handle der Quelldatei    */
HFILE hfDest        /*  Handle der Zieldatei     */
```

Beschreibung

Diese Funktion kopiert eine Quelldatei in eine Zieldatei. Falls die Quelldatei mit dem Microsoft-Programm COMPRESS.EXE komprimiert wurde, erstellt die Funktion eine entkomprimierte Zieldatei. War die Quelldatei nicht komprimiert, kopiert die Funktion einfach die Orginaldatei.

Parameter

hfSource: Bezeichnet die Quelldatei. Dieses Handle ist der Rückgabewert der Funktion *LZOpenFile* beim Öffnen einer komprimierten Datei.

hfDest: Bezeichnet die Zieldatei.

Rückgabewert

Ist die Funktion erfolgreich, gibt sie die Größe der Zieldatei in Byte zurück. Andernfalls liefert sie einen Fehlerwert kleiner Null. Folgende Werte sind möglich:

Wert	Bedeutung
LZERROR_BADINHANDLE	Das Handle, das die Zieldatei bezeichnet, war nicht gültig.
LZERROR_BADOUTHANDLE	Das Handle, das die Quelldatei bezeichnet, war nicht gültig.
LZERROR_GLOBALLOC	Für die benötigten Puffer steht nicht genügend Speicher zu Verfügung.
LZERROR_GLOBLOCK	Das Handle, das die internen Datenstrukturen bezeichnet, ist nicht gültig.
LZERROR_READ	Das Format der Quelldatei ist ungültig.
LZERROR_UNKNOWNALG	Die Quelldatei ist mit einem unbekannten Algorithmus komprimiert worden.
LZERROR_WRITE	Es steht nicht genügend Platz für die Ausgabedatei zur Verfügung.

Hinweis

Diese Funktion ist für Operationen konzipiert, bei denen nur eine Datei kopiert wird. Sollen mehrere Dateien kopiert werden, steht die Funktion *CopyLZFile* zur Verfügung.

Ist die Funktion erfolgreich, wird die durch *hfDest* bezeichnete Datei entkomprimiert.

Falls die Quell- oder die Zieldatei mit einer C-Standardfunktion geöffnet wird (anstelle der Funktionen *_lopen* oder *OpenFile*), muß sie im Binärmodus geöffnet werden.

Beispiel — Das folgende Beispiel benutzt die Funktion *LZCopy* zum Kopieren einer Datei:

```
char szSrc[] = {"readme.txt"};
char szDst[] = {"readme.bak"};
OFSTRUCT ofStrSrc;
OFSTRUCT ofStrDest;
HFILE hfSrcFile, hfDstFile;

/* öffnet die Quelldatei */

hfSrcFile = LZOpenFile(szSrc, &ofStrSrc, OF_READ);

/* Erzeugt die Zieldatei */

hfDstFile = LZOpenFile(szDst, &ofStrDest, OF_CREATE);

/* Kopiert die Quelldatei zur Zieldatei */

LZCopy(hfSrcFile, hfDstFile);

/* Schließt die Dateien */

LZClose(hfSrcFile);
LZClose(hfDstFile);
```

Siehe auch — *CopyLZFile, _lopen, LZOpenFile, OpenFile*

LZDone 3.1

Syntax

```
#include <lzexpand.h>
void LZDone(void)
```

Beschreibung — *LZDone* gibt Puffer frei, die die Funktion *LZStart* für Operationen reserviert hat, bei denen mehrere Dateien kopiert werden.

Parameter — Diese Funktion besitzt keine Parameter.

Rückgabewert — Diese Funktion hat keinen Rückgabewert.

Hinweis — Anwendungen, die mehrere Dateien kopieren, sollten zunächst die Funktion *LZStart* aufrufen, bevor die Dateien mit *CopyLZFile* kopiert werden. *LZStart* reserviert Puffer für die Kopieroperationen.

Beispiel — Das folgende Beispiel ruft *LZDone* auf, um die durch *LZStart* reservierten Puffer freizugeben:

```
#define NUM_FILES   4

char *szSrc[NUM_FILES] =
    {"readme.txt", "data.txt", "update.txt",
     "list.txt"};
char *szDest[NUM_FILES] =
    {"readme.bak", "data.bak", "update.bak",
     "list.bak"};
```

```
OFSTRUCT ofStrSrc;
OFSTRUCT ofStrDest;
HFILE hfSrcFile, hfDstFile;
int i;

/* Reserviert interne Puffer für CopyLZFile */

LZStart();

/* ¹ffnet, kopiert und schließt die Dateien */

for (i = 0; i  NUM_FILES; i++) {
   hfSrcFile = LZOpenFile(szSrc[i], &ofStrSrc, OF_READ);
   hfDstFile = LZOpenFile(szDest[i], &ofStrDest,
                          OF_CREATE);
   CopyLZFile(hfSrcFile, hfDstFile);
   LZClose(hfSrcFile);
   LZClose(hfDstFile);
}

LZDone(); /* Gibt die internen Puffer frei */
```

Siehe auch *CopyLZFile, LZCopy, LZStart*

LZInit 3.1

Syntax

```
#include <lzexpand.h>

HFILE LZInit(hfSrc)

HFILE hfSrc;  /*  Handle der Quelldatei  */
```

Beschreibung Diese Funktion reserviert Speicher für die internen Datenstrukturen, die zum Entkomprimieren einer Datei benötigt werden, erzeugt diese Strukturen und initialisiert sie.

Parameter *hfSrc:* Bezeichnet die Quelldatei.

Rückgabewert Der Rückgabewert ist das Handle der Orginaldatei, wenn die Funktion erfolgreich und die Datei nicht komprimiert ist. Ist die Funktion erfolgreich und die Datei komprimiert, so ist der Rückgabewert ein neues Datei-Handle. Schlägt die Funktion fehl, so ist der Rückgabewert ein Fehlerwert kleiner Null. Folgende Werte sind möglich:

Wert	Bedeutung
LZERROR_BADIN-HANDLE	Das Handle, das die Quelldatei bezeichnet, ist ungültig.
LZERROR_GLOB-ALLOC	Für die internen Datenstrukturen steht nicht genügend Speicher zur Verfügung. Die Funktion liefert diesen Wert, falls eine Anwendung mehr als 16 Dateien zu öffnen versucht.

Wert	Bedeutung
LZERROR_GLOBLOCK	Das Handle, das den globalen Speicher bezeichnet, ist ungültig. Der interne Aufruf der Funktion *GlobalLock* ist fehlgeschlagen.
LZERROR_READ	Das Format der Quelldatei ist ungültig.
LZERROR_UNKNOWNALG	Die Datei wurde mit einem unbekannten Algorithmus komprimiert.

Hinweis

Es können maximal 16 komprimierte Dateien gleichzeitig geöffnet sein.

Beispiel

Das folgende Beispiel ruft *LZInit* auf, um die internen Strukturen zu initialisieren, die zum Entkomprimieren von Dateien benötigt werden:

```
char szSrc[] = {"readme.cmp"};
char szFileName[128];
OFSTRUCT ofStrSrc;
OFSTRUCT ofStrDest;
HFILE hfSrcFile, hfDstFile, hfCompFile;
int cbRead;
BYTE abBuf[512];

/* Öffnet die komprimierte Quelldatei */

hfSrcFile = OpenFile(szSrc, &ofStrSrc, OF_READ);

/*
 * Initialisiert die internen Datenstrukturen für die
 * Entkomprimierung
 */

hfCompFile = LZInit(hfSrcFile);

/* Ermittelt den ursprünglichen Namen der
   komprimierten Datei */

GetExpandedName(szSrc, szFileName);

/* Erzeugt die Zieldatei mit dem ursprünglichen
   Namen */

hfDstFile = LZOpenFile(szFileName, &ofStrDest,
                       OF_CREATE);

/* Kopiert die komprimierte Quelldatei
   zur Zieldatei */

do {
    if ((cbRead = LZRead(hfCompFile, abBuf,
         sizeof(abBuf))) > 0)
       _lwrite(hfDstFile, abBuf, cbRead);
```

```
    else {
        .
        . /* Hier sollten die Fehlerbedingungen
        .    bearbeitet werden */
        .
    }
} while (cbRead == sizeof(abBuf));

/* Schließt die Dateien */

LZClose(hfSrcFile);
LZClose(hfDstFile);
```

LZOpenFile 3.1

Syntax

```
#include <lzexpand.h>

HFILE LZOpenFile(lpszFile, lpof, style)

LPCSTR lpszFile;     /*  Adresse des Dateinamens      */
OFSTRUCT FAR* lpof;  /*  Adresse der Struktur für die
                         Dateiinformationen           */
UINT style;          /*  Aktion, die ausgeführt
                         werden soll                  */
```

Beschreibung Diese Funktion erzeugt, löscht oder öffnet die Datei, die durch den String bestimmt wird, auf den *lpszFile* zeigt.

Parameter *lpszFile:* Zeigt auf einen String, der den Dateinamen angibt.

lpof: Zeigt auf eine *OFSTRUCT*-Struktur, die beim Öffnen der Datei die Informationen über die Datei empfängt. Die Struktur kann in nachfolgenden Aufrufen von *LZOpenFile* verwendet werden, um sich auf die geöffnete Datei zu beziehen.

Das Element *szPathName* dieser Struktur enthält Zeichen aus dem OEM-Zeichensatz.

style: Bestimmt die Aktion, die ausgeführt werden soll. Die Werte können mit dem bitweisen ODER-Operator kombiniert werden:

Wert	Bedeutung
OF_CANCEL	Fügt einen Abbrechen-Schalter in das Dialogfenster *OF_PROMPT* ein. Wird dieser Schalter ausgewählt, liefert *LZOpenFile* die Fehlermeldung `Datei nicht gefunden`.
OF_CREATE	Weist *LZOpenFile* an, eine neue Datei zu erzeugen. Falls die Datei schon existiert, wird sie auf die Länge Null gekürzt.
OF_DELETE	Löscht die Datei.

Wert	Bedeutung
OF_EXIST	Öffnet die Datei und schließt sie dann wieder. Diese Aktion dient als Test auf die Existenz einer Datei.
OF_PARSE	Füllt die *OFSTRUCT*-Struktur, führt aber keine weitere Aktion aus.
OF_PROMPT	Zeigt ein Dialogfenster an, wenn die angeforderte Datei nicht existiert. Das Dialogfenster zeigt dem Benutzer, daß Windows die Datei nicht finden kann und fordert ihn auf, die Diskette mit der Datei in Laufwerk A einzulegen.
OF_READ	Öffnet die Datei nur zum Lesen.
OF_READWRITE	Öffnet die Datei zum Lesen und Schreiben.
OF_REOPEN	Öffnet die Datei unter Verwendung der Informationen in der *OFSTRUCT*-Struktur, auf die der Parameter *lpof* zeigt.
OF_SHARE_-DENY_NONE	Öffnet die Datei, ohne anderen Programmen den Zugriff zum Lesen oder Schreiben aus bzw. in die Datei zu verweigern. *LZOpenFile* kann nicht erfolgreich durchgeführt werden, falls die Datei durch ein anderes Programm im Kompatibilitätsmodus geöffnet wurde.
OF_SHARE_-DENY_READ	Öffnet die Datei und verweigert anderen Programmen den Lesezugriff auf die Datei. *LZOpenFile* kann nicht erfolgreich durchgeführt werden, falls die Datei durch ein anderes Programm im Kompatibilitätsmodus oder für Lesezugriff geöffnet wurde.
OF_SHARE_-DENY_WRITE	Öffnet die Datei und verweigert anderen Programmen den Schreibzugriff auf die Datei. *LZOpenFile* kann nicht erfolgreich durchgeführt werden, falls die Datei durch ein anderes Programm im Kompatibilitätsmodus oder für Schreibzugriff geöffnet wurde.
OF_SHARE_-EXCLUSIVE	Öffnet die Datei im Exklusivmodus. Anderen Programmen wird sowohl der Lese- als auch der Schreibzugriff auf die Datei verweigert. *LZOpenFile* schlägt fehl, wenn die Datei in einem anderen Modus zum Lesen oder Schreiben geöffnet wurde, auch wenn dies durch das laufende Programm geschehen ist.
OF_WRITE	Öffnet die Datei nur zum Schreiben.

Rückgabewert Der Rückgabewert ist ein Handle, das die Datei bezeichnet, wenn die Funktion erfolgreich und der in *style* angegebene Wert nicht *OF_READ* ist. Wird eine komprimierte Datei geöffnet, während *style* auf *OF_READ* gesetzt ist, ist der Rückgabewert ein spezielles Datei-Handle.

Kann die Funktion nicht erfolgreich durchgeführt werden, ist der Rückgabewert -1.

Hinweis Hat *style* den Wert *OF_READ* (oder *OF_READ* und eines der *OF_SHARE_*-Flags) und ist die Datei komprimiert, ruft *LZOpenFile* die Funktion *LZInit* auf, die die nötige Initialisierung für die Entkomprimierung vornimmt.

Beispiel Das folgende Beispiel ruft *LZOpenFile* auf, um die Quelldatei zu öffnen und eine Zieldatei zu erstellen, in die die Quelldatei kopiert werden kann.

```
char szSrc[] = {"readme.txt"};
char szDst[] = {"readme.bak"};
OFSTRUCT ofStrSrc;
OFSTRUCT ofStrDest;
HFILE hfSrcFile, hfDstFile;

/* ffnet die Quelldatei */

hfSrcFile = LZOpenFile(szSrc, &ofStrSrc, OF_READ);

/* Erzeugt die Zieldatei */

hfDstFile = LZOpenFile(szDst, &ofStrDest, OF_CREATE);

/* Kopiert die Quelldatei zur Zieldatei */

LZCopy(hfSrcFile, hfDstFile);

/* Schließt die Dateien */

LZClose(hfSrcFile);
LZClose(hfDstFile);
```

Siehe auch *LZInit*

LZRead 3.1

Syntax

```
#include <lzexpand.h>
int LZRead(hf, lpvBuf, cb)

HFILE hf;          /* Handle der Datei                */
void FAR lpvBuf;   /* Adresse des Puffers für
                      die Dateidaten                  */
int cb;            /* Anzahl der einzulesenden Byte   */
```

Beschreibung Diese Funktion liest Byte aus einer Datei in einen Puffer.

Parameter *hf:* Bezeichnet die Quelldatei.

lpvBuf: Zeigt auf einen Puffer, der die aus der Datei eingelesenen Byte aufnimmt.

ch: Bestimmt die maximale Anzahl der zu lesenden Byte.

Rückgabewert Ist die Funktion erfolgreich, so ist der Rückgabewert die tatsächlich eingelesene Anzahl Byte, andernfalls ist er kleiner als 0. Es können folgende Werte auftreten:

Wert	Bedeutung
LZERROR_BADINHANDLE	Das Handle, das die Quelldatei kennzeichnet, ist ungültig.
LZERROR_BADVALUE	Der Parameter *cb* enthält einen negativen Wert.
LZERROR_GLOBLOCK	Das Handle, das die nötigen Initialisierungsdaten bezeichnet, ist ungültig.
LZERROR_READ	Das Format der Quelldatei ist ungültig.
LZERROR_UNKNOWNALG	Die Datei wurde mit einem unbekannten Algorithmus komprimiert.

Hinweis Falls die Datei nicht komprimiert ist, ruft *LZRead* die Funktion *_lread* auf, um die Leseoperation durchzuführen.

Ist die Datei komprimiert, emuliert *LZRead* die Funktion *_lread* für eine entkomprimierte Version der Datei und kopiert die Datenbyte in den Puffer, auf den *lpvBuf* zeigt.

Falls die Quelldatei mit dem Microsoft-Programm COMPRESS.EXE komprimiert wurde, können anstelle der Funktionen *OpenFile*, *_llseek* und *_lread* die Funktionen *LZOpenFile*, *LZSeek* und *LZRead* aufgerufen werden.

Beispiel Das folgende Beispiel verwendet *LZRead*, um eine komprimierte Datei zu kopieren und entkomprimieren:

```
char szSrc[] = {"readme.cmp"};
char szFileName[128];
OFSTRUCT ofStrSrc;
OFSTRUCT ofStrDest;
HFILE hfSrcFile, hfDstFile, hfCompFile;
int cbRead;
BYTE abBuf[512];

/* ¹ffnet die komprimierte Quelldatei */

hfSrcFile = OpenFile(szSrc, &ofStrSrc, OF_READ);

/* Initialisiert die internen Datenstrukturen für die
 * Entkomprimierung  */

hfCompFile = LZInit(hfSrcFile);
```

```
/* Ermittelt den ursprünglichen Namen der
   komprimierten Datei */

GetExpandedName(szSrc, szFileName);

/* Erzeugt die Zieldatei mit dem ursprünglichen
   Namen */

hfDstFile = LZOpenFile(szFileName, &ofStrDest,
                       OF_CREATE);

/* Kopiert die komprimierte Quelldatei zur
   Zieldatei */

do {
    if ((cbRead = LZRead(hfCompFile, abBuf,
         sizeof(abBuf))) > 0)
       _lwrite (hfDstFile, abBuf, cbRead);
    else {
        .
        . /* Hier sollten die Fehlerbedingungen
             bearbeitet werden */
        .
    }
} while (cbRead == sizeof(abBuf));

/* Schließt die Dateien. */

LZClose(hfSrcFile);
LZClose(hfDstFile);
```

Siehe auch *_llseek, _lread, LZOpenFile, LZRead, LZSeek*

LZSeek 3.1

Syntax

```
#include <lzexpand.h>

LONG LZSeek(hf, lOffset, nOrigin)

HFILE hf;        /* Handle der Datei                */
long lOffset;    /* Anzahl der zu verschiebenden
                    Byte                            */
int nOrigin;     /* ursprüngliche Position          */
```

Beschreibung Diese Funktion bewegt einen Dateizeiger von der Ausgangsposition auf eine neue Position.

Parameter *hf:* Bezeichnet die Quelldatei.

lOffset: Bestimmt die Anzahl Byte, um die der Dateizeiger verschoben werden soll.

nOrigin: Bestimmt die Ausgangsposition des Zeigers. Dieser Parameter muß einen der folgenden Werte haben:

Wert	Bedeutung
0	Bewegt den Dateizeiger *lOffset* Byte vom Dateianfang aus.
1	Bewegt den Dateizeiger *lOffset* Byte von der aktuellen Position aus.
2	Bewegt den Dateizeiger *lOffset* Byte vom Dateiende aus.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert der Offset vom Dateianfang zur neuen Dateizeigerposition. Andernfalls liefert die Funktion einen Fehlerwert kleiner Null. Es können folgende Fehler auftreten:

Wert	Bedeutung
LZERROR_BAD-INHANDLE	Das Handle, das die Quelldatei bezeichnet, war ungültig.
LZERROR_BAD-VALUE	Einer der Parameter überschreitet den erlaubten Wertebereich.
LZERROR_-GLOBLOCK	Das Handle, das die Initialisierungsdaten bezeichnet, ist ungültig.

Hinweis Ist die Datei nicht komprimiert, ruft *LZSeek* die Funktion *_llseek* auf und verschiebt den Dateizeiger um den angegebenen Offset.

Falls die Datei komprimiert ist, emuliert *LZSeek* die Funktion *_llseek* für eine entkomprimierte Version der Datei

Siehe auch *_llseek*

LZStart 3.1

Syntax
```
#include <lzexpand.h>

int LZStart(void)
```

Beschreibung Diese Funktion reserviert die Puffer, die die Funktion *LZCopyFile* benutzt, um eine Quelldatei in eine Zieldatei zu kopieren.

Parameter Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich ist. Sonst ist er *LZERROR_GLOBALLOC*.

Hinweis Anwendungen, die mehrere aufeinanderfolgende Dateien kopieren oder kopieren und entkomprimieren, sollten die Funktionen *LZStart*, *CopyLZFile* und *LZDone* aufrufen. Anwendungen, die einzelne Dateien kopieren, sollten die Funktion *LZCopy* aufrufen.

Beispiel

Das folgende Beispiel verwendet *LZStart* um die von *CopyLZFile* benötigten Puffer zu reservieren:

```
#define NUM_FILES   4

char *szSrc[NUM_FILES] =
    {"readme.txt", "data.txt", "update.txt",
     "list.txt"};
char *szDest[NUM_FILES] =
    {"readme.bak", "data.bak", "update.bak",
     "list.bak"};
OFSTRUCT ofStrSrc;
OFSTRUCT ofStrDest;
HFILE hfSrcFile, hfDstFile;
int i;

/* Reserviert interne Puffer für die
   Funktion CopyLZFile */

LZStart();

/* ¹ffnet, kopiert und schließt die Dateien */

for (i = 0; i < NUM_FILES; i++) {
    hfScrFile = LZOpenFile(szSrc[i], &ofStrSrc,
                           OF_READ);
    hfDstFile = LZOpenFile(szDest[i], &ofStrDest,
                OF_CREATE);
    CopyLZFile(hfSrcFile, hfDstFile);
    LZClose(hfSrcFile);
    LZClose(hfDstFile);
}

LZDone(); /* Gibt die internen Puffer frei */
```

Siehe auch

CopyLZFile, LZCopy, LZDone

MapWindowPoints 3.1

Syntax

```
void MapWindowPoints(hwndFrom, hwndTo, lppt, cPoints)

HWND hwndFrom;   /* Handle des Fensters, auf das die
                    Koordinaten bezogen sind          */
HWND hwndTo;     /* Handle des Fensters, auf das Bezug
                    genommen werden soll              */
POINT FAR lppt;  /* Adresse des Struktur-Array mit den
                    zu konvertierenden Punkten        */
UINT cPoints;    /* Anzahl der Strukturen im Array    */
```

Beschreibung

Diese Funktion konvertiert eine Reihe von Punkten aus einem auf ein bestimmtes Fenster bezogenem Koordinatensystem in ein anderes, auf ein anderes Fenster bezogenes Koordinatensystem.

Parameter

hwndFrom: Bezeichnet das Fenster, aus dem die Punkte konvertiert werden. Ist dieser Parameter *NULL* oder *HWND_DESKTOP*, wird angenommen, daß die Punkte in Bidschirmkoordinaten gegeben sind.

hwndTo: Bezeichnet das Fenster, auf das die Koordinaten bezogen werden. Falls dieser Parameter *NULL* oder *HWND_DESKTOP* ist, werden die Punkte in Bildschirmkoordinaten konvertiert.

lppt: Zeigt auf ein Array von *POINT*-Strukturen, das die zu konvertierenden Punkte enthält. Dieser Parameter kann auch auf eine *RECT*-Struktur zeigen. In diesem Fall sollte der Parameter *cPoints* auf 2 gesetzt werden. Die *POINT*-Struktur hat folgende Form:

```
typedef struct tagPOINT { /* pt */
    int x;
    int y;
} POINT;
```

Die *RECT*-Struktur hat folgende Form:

```
typedef struct tagRECT { /* rc */
    int left;
    int top;
    int right;
    int bottom;
} RECT;
```

cPoints: Bestimmt die Anzahl der *POINT*-Strukturen in dem durch den Parameter *lppt* bezeichneten Array.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Siehe auch *ClientToScreen*, *ScreenToClient*

MemManInfo 3.1

Syntax

```
#include <toolhelp.h>
BOOL MemManInfo(lpmmi)

MEMMANINFO FAR* lpmmi;  /* Adresse der
                           MEMMANINFO-Struktur       */
```

Beschreibung Diese Funktion füllt die angegebene Struktur mit Informationen über den Status und die Leistungsfähigkeit der Speicherverwaltung. Diese Funktion ist im erweiterten 386-Modus besonders hilfreich, kann aber auch im Standardmodus verwendet werden.

Parameter *lpmmi:* Zeigt auf eine *MEMMANINFO*-Struktur, die die Informationen über die Speicherverwaltung empfängt.

Die *MEMANINFO*-Struktur hat folgende Form:

```
#include <toolhelp.h>

typedef struct tagMEMMANINFO {  /* mmi */
    DWORD dwSize;
    DWORD dwLargestFreeBlock;
    DWORD dwMaxPagesAvailable;
    DWORD dwMaxPagesLockable;
    DWORD dwTotalLinearSpace;
    DWORD dwTotalUnlockedPages;
    DWORD dwFreePages;
    DWORD dwTotalPages;
    DWORD dwFreeLinearSpace;
    DWORD dwSwapFilePages;
    WORD wPageSize;
} MEMMANINFO;
```

Rückgabewert Ist die Funktion erfolgreich, so ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Diese Funktion dient Informationszwecken.

Vor einem Aufruf von *MemManInfo* muß eine Anwendung die *MEMMANINFO*-Struktur initialisieren und im Element *dwSize* deren Größe in Byte angeben.

MemoryRead 3.1

Syntax

```
#include <toolhelp.h>

DWORD MemoryRead(wSel, dwOffset, lpvBuf, dwcb)

WORD wSel;          /* Selektor des globalen
                       Heap-Objekts                 */
DWORD dwOffset;     /* Offset des Objekts           */
void FAR* lpvBuf;   /* Adresse des Puffers, in den
                       kopiert werden soll          */
DWORD dwcb;         /* Anzahl der zu lesenden Byte  */
```

Beschreibung Diese Funktion kopiert Speicher aus dem angegebenen globalen Heap-Objekt in den angegebenen Puffer.

Parameter *wSel:* Bestimmt das globale Heap-Objekt, aus dem gelesen werden soll. Dieser Wert muß ein Selektor auf dem globalen Heap sein. Ist der Wert ein Alias-Selektor oder ein Selektor in ein Array mit aufgespaltenen Selektoren, schlägt die Funktion *MemoryRead* fehl.

dwOffset: Bestimmt den Offset des durch *wSel* angegebenen Objekts, an dem das Lesen beginnt. Der Wert *dwOffset* kann auf eine beliebige Stelle im Objekt zeigen. Er darf auch größer als 64 KByte sein, wenn das Objekt größer als 64 KByte ist.

lpvBuf: Zeigt auf den Puffer, in den *MemoryRead* den Speicher aus dem Objekt kopieren soll. Dieser Puffer muß groß genug sein, um den gesamten kopierten Speicher aufnehmen zu können. Steht für die Anwendung nur noch wenig freier Speicher zur Verfügung, sollte sich *lpvBuf* in einem festen Objekt befinden, während *MemoryRead* die Daten kopiert.

dwcb: Bestimmt die Anzahl Byte, die aus dem Objekt in den durch *lpvBuf* bezeichneten Puffer kopiert werden sollen.

Rückgabewert Der Rückgabewert ist die Anzahl Byte, die aus *wSel* nach *lpvBuf* kopiert worden sind. Ist *wSel* ungültig oder überschreitet *dwOffset* den Bereich des Selektors, ist der Rückgabewert Null.

Hinweis Die Funktion *MemoryRead* erlaubt Programmierern Speicher zu untersuchen, ohne auf Alias-Selektoren oder aufgespaltene Selektoren Rücksicht nehmen zu müssen. *MemoryRead* liest Speicher in Objekten zum Lesen und Schreiben oder in solchen, die nur gelesen werden können. Die Funktion kann auf Objekte beliebiger Größe angewendet werden, egal zu welcher Task sie gehören. Die Offsets des Selektor-Array müssen nicht berechnet werden.

Die Funktionen *MemoryRead* und *MemoryWrite* sind dafür konzipiert, Objekte zu lesen und zu schreiben, die mit der Funktion *LoadModule* geladen oder mit der Funktion *GlobalAlloc* reserviert wurden. Programmierer sollten den Selektoranteil eines **far**-Zeigers nicht abspalten, um ihn als Wert für *wSel* zu benutzen, außer wenn feststeht, daß der Selektor sich auf dem globalen Heap befindet.

Siehe auch *MemoryWrite*

MemoryWrite 3.1

Syntax

```
#include <toolhelp.h>

DWORD MemoryWrite(wSel, dwOffset, lpvBuf, dwcb)

WORD wSel;          /* Selektor des globalen
                       Heap-Objekts                */
DWORD dwOffset;     /* Offset des Objekts          */
void FAR* lpvBuf;   /* Adresse des Puffers, aus dem
                       geschrieben werden soll     */
DWORD dwcb;         /* Anzahl der zu schreibenden
                       Byte                        */
```

Beschreibung Diese Funktion kopiert Speicher aus dem angegebenen Puffer in ein bestimmtes globales Heap-Objekt.

Parameter *wSel:* Bestimmt das globale Heap-Objekt, in das geschrieben werden soll. Dieser Wert muß ein Selektor auf dem globalen Heap sein. Ist der Wert ein Alias-Selektor oder ein Selektor in

ein Array mit aufgespaltenen Selektoren, schlägt die Funktion *MemoryWrite* fehl.

dwOffset: Bestimmt den Offset im Objekt, an dem mit dem Schreiben begonnen werden soll. Der Wert *dwOffset* kann auf eine beliebige Stelle im Objekt zeigen. Er darf auch größer als 64 KByte sein, wenn das Objekt größer als 64 KByte ist.

lpvBuf: Zeigt auf den Puffer, aus dem *MemoryWrite* den Speicher in das Objekt kopieren soll. Steht für die Anwendung nur noch wenig freier Speicher zur Verfügung, sollte sich *lpvBuf* in einem festen Objekt befinden, während *MemoryWrite* die Daten kopiert.

dwcb: Bestimmt die Anzahl Byte, die aus dem durch *lpvBuf* bezeichneten Puffer in das Objekt kopiert werden sollen.

Rückgabewert Der Rückgabewert ist die Anzahl Byte, die aus *lpvBuf* nach *wSel* kopiert wurden. Ist der Selektor ungültig oder überschreitet *dwOffset* den Bereich des Selektors, so ist der Rückgabewert Null.

Hinweis Die Funktion *MemoryWrite* erlaubt Programmierern Speicherinhalte zu verändern, ohne auf Alias-Selektoren oder aufgespaltene Selektoren Rücksicht nehmen zu müssen. *MemoryWrite* beschreibt Speicher in Objekten zum Lesen und Schreiben oder in solchen, die nur gelesen werden können. Die Funktion kann auf Objekte beliebiger Größe angewendet werden, die einer beliebigen Task zugehörig sein können. Der Offset des Selektor-Array muß nicht berechnet werden. Alias-Objekte müssen nicht beschreibbar gemacht werden.

Die Funktionen *MemoryRead* und *MemoryWrite* sind dafür konzipiert, Objekte zu lesen und zu schreiben, die mit der Funktion *LoadModule* geladen oder mit der Funktion *GlobalAlloc* reserviert worden sind. Programmierer sollten den Selektoranteil eines **far**-Zeigers nicht abspalten, um ihn als Wert für *wSel* zu benützen, außer wenn feststeht, daß der Selektor sich auf dem globalen Heap befindet.

Siehe auch *MemoryRead*

MessageProc 3.1

Syntax

```
LRESULT CALLBACK MessageProc(code, wParam, lParam)

int code;          /* Botschaftstyp                */
WPARAM wParam;     /* nicht definiert              */
LPARAM lParam;     /* Adresse der Struktur mit den
                      Botschaftsdaten              */
```

Beschreibung Die Funktion *MessageProc* ist eine anwendungs- oder bibliotheksdefinierte Callback-Funktion, die vom System aufgerufen

wird, nachdem ein Dialogfenster, ein Meldungsfenster oder ein Menü eine Botschaft abgerufen hat. *MessageProc* wird aufgerufen, bevor die Botschaft verarbeitet wird. Die Callback-Funktion kann die Botschaften bearbeiten oder ändern.

Parameter

code: Bestimmt den Typ der zu bearbeitenden Botschaft. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
MSGF_DIALOGBOX	Botschaften innerhalb einer Dialogfenster- oder Meldungsfensterfunktion werden bearbeitet.
MSGF_MENU	Tastatur- und Mausbotschaften in einem Menü werden bearbeitet.

Wenn der Parameter *code* kleiner als Null ist, muß die Callback-Funktion die Botschaft ohne weitere Bearbeitung an *CallNextHookEx* übergeben und den von dieser Funktion zurückgegebenen Wert zurückliefern.

wParam: Bestimmt einen *NULL*-Wert.

lParam: Zeigt auf eine *MSG*-Struktur. Die *MSG*-Struktur hat folgende Form:

```
typedef struct tagMSG {   /* msg */
   HWND hwnd;
   UINT message;
   WPARAM wParam;
   LPARAM lParam;
   DWORD time;
   POINT pt;
} MSG;
```

Rückgabewert

Die Callback-Funktion sollte einen Wert ungleich Null zurückgeben, wenn sie die Botschaft bearbeitet. Sie sollte den Wert Null zurückgeben, wenn sie die Botschaft nicht bearbeitet.

Hinweis

Der Filtertyp *WH_MSGFILTER* ist der einzige task-spezifische Filter. Eine Task kann diesen Filter installieren.

Anwendungen müssen die Callback-Funktion installieren, indem sie den Filtertyp *WH_MSGFILTER* und die Adresse der Prozedurinstanz der Callback-Funktion in einem Aufruf der Funktion *SetWindowsHookEx* übergeben.

MessageProc ist ein Platzhalter für den bibliotheksdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Bibliothek exportiert werden.

Siehe auch

CallNextHookEx, SetWindowsHookEx

ModuleFindHandle 3.1

Syntax

```
#include <toolhelp.h>
HMODULE ModuleFindHandle(lpme, hmod)

HMODULEENTRY FAR* lpme;   /* Adresse der MODULEENTRY-
                             Struktur                */
HMODULE hmod;             /* Handle des Moduls       */
```

Beschreibung Die Funktion *ModuleFindHandle* füllt die angegebene Struktur mit Informationen, die das angegebene Modul beschreiben.

Parameter *lpme:* Zeigt auf eine *MODULEENTRY*-Struktur, die Informationen über das Modul aufnimmt. Die *MODULEENTRY*-Struktur hat folgende Form:

```
#include <toolhelp.h>
typedef struct tagMODULEENTRY {   /* me */
    DWORD dwSize;
    char szModule[MAX_MODULE_NAME + 1];
    HMODULE hModule;
    WORD wcUsage;
    char szExePath[MAX_PATH + 1];
    WORD wNext;
} MODULEENTRY;
```

hmod: Bezeichnet das zu beschreibende Modul.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert das Handle des angegebenen Moduls. Andernfalls ist der Rückgabewert *NULL*.

Hinweis *ModuleFindHandle* gibt Informationen über ein aktuell geladenes Modul zurück, dessen Modul-Handle bekannt ist.

Diese Funktion kann benutzt werden, um eine Überprüfung der Liste aller aktuell geladenen Module zu beginnen. Anwendungen können folgende Einträge in der Modulliste untersuchen, indem sie die Funktion *ModuleNext* aufrufen.

Anwendungen müssen die *MODULEENTRY*-Struktur initialisieren und ihre Größe im Element *dwSize* in Byte angeben, bevor sie *ModuleFindHandle* aufrufen.

Siehe auch *ModuleFindName, ModuleFirst, ModuleNext*

ModuleFindName 3.1

Syntax

```
#include <toolhelp.h>
HMODULE ModuleFindName(lpme, lpszName)

MODULEENTRY FAR* lpme;  /* Adresse der MODULEENTRY-
                           Struktur                 */
LPCSTR lpszName;        /* Adresse des Modulnamens  */
```

Beschreibung Die Funktion *ModuleFindName* füllt die angegebene Struktur mit Informationen, die das Modul mit dem angegebenen Namen beschreiben.

Parameter *lpme:* Zeigt auf eine *MODULEENTRY*-Struktur, die Informationen über das Modul aufnimmt. Die *MODULEENTRY*-Struktur hat folgende Form:

```
#include <toolhelp.h>

typedef struct tagMODULEENTRY {   /* me */
    DWORD dwSize;
    char szModule[MAX_MODULE_NAME + 1];
    HMODULE hModule;
    WORD wcUsage;
    char szExePath[MAX_PATH + 1];
    WORD wNext;
} MODULEENTRY;
```

lpszName: Bestimmt den Namen des zu beschreibenden Moduls.

Rückgabewert Der Rückgabewert ist das Handle des im Parameter *lpszName* benannten Moduls. Schlägt die Funktion fehl, ist der Rückgabewert *NULL*.

Hinweis Die Funktion *ModuleFindName* gibt Informationen über ein aktuell geladenes Modul zurück, indem sie den Namen des Moduls in der Modulliste sucht.

Diese Funktion kann zu Beginn einer Überprüfung der Liste aller aktuell geladenen Module benutzt werden. Anwendungen können folgende Einträge in der Modulliste untersuchen, indem sie die Funktion *ModuleNext* aufrufen.

Anwendungen müssen die *MODULEENTRY*-Struktur initialisieren und ihre Größe im Element *dwSize* in Byte angeben, bevor sie *ModuleFindName* aufrufen.

Siehe auch *ModuleFindHandle*, *ModuleFirst*, *ModuleNext*

ModuleFirst 3.1

Syntax

```
#include <toolhelp.h>
BOOL ModuleFirst(lpme)

MODULEENTRY FAR* lpme;   /* Adresse der MODULEENTRY-
                            Struktur                */
```

Beschreibung Die Funktion *ModuleFirst* füllt die angegebene Struktur mit Informationen, die das erste Modul in der Liste aller aktuell geladenen Module beschreiben.

Parameter *lpme:* Zeigt auf eine *MODULEENTRY*-Struktur, die Informationen über das Modul aufnimmt.

Die *MODULEENTRY*-Struktur hat folgende Form:

```
#include <toolhelp.h>

typedef struct tagMODULEENTRY {   /* me */
    DWORD dwSize;
    char szModule[MAX_MODULE_NAME + 1];
    HMODULE hModule;
    WORD wcUsage;
    char szExePath[MAX_PATH + 1];
    WORD wNext;
} MODULEENTRY;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *ModuleFirst* kann zum Beginn einer Überprüfung der Liste aller aktuell geladenen Module benutzt werden. Anwendungen können folgende Einträge in der Modulliste durch Aufruf der Funktion *ModuleNext* untersuchen.

Anwendungen müssen die *MODULEENTRY*-Struktur initialisieren und ihre Größe im Element *dwSize* in Byte angeben, bevor sie *ModuleFirst* aufrufen.

Siehe auch *ModuleFindHandle, ModuleFindName, ModuleNext*

ModuleNext 3.1

Syntax

```
#include<toolhelp.h>
BOOL ModuleNext(lpme)

MODULEENTRY FAR* lpme;   /* Adresse der MODULEENTRY-
                            Struktur                */
```

Beschreibung Die Funktion *ModuleNext* füllt die angegebene Struktur mit Informationen, die das folgende Modul in der Liste aller aktuell geladenen Module beschreiben.

Parameter *lpme:* Zeigt auf eine *MODULEENTRY*-Struktur, die Informationen über das Modul aufnimmt. Die *MODULEENTRY*-Struktur hat folgende Form:

```
#include <toolhelp.h>

typedef struct tagMODULEENTRY {   /* me */
    DWORD dwSize;
    char szModule[MAX_MODULE_NAME + 1];
    HMODULE hModule;
    WORD wcUsage;
    char szExePath[MAX_PATH + 1];
    WORD wNext;
} MODULEENTRY;
```

Rückgabewert Ist die Funktion erfolgreich, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *ModuleNext* kann zum Fortsetzen einer Überprüfung der Liste aller aktuell geladenen Module benutzt werden. Die Suche muß mit *ModuleFirst*, *ModuleFindName* oder *ModuleFindHandle* begonnen werden.

Siehe auch *ModuleFindHandle*, *ModuleFindName*, *ModuleFirst*

MouseProc 3.1

Syntax

```
LRESULT CALLBACK MouseProc(code, wParam, lParam)

int code;        /* Botschaftsbearbeitungs-Flag       */
WPARAM wParam;   /* Botschaftsbezeichner              */
LPARAM lParam;   /* Adresse der MOUSEHOOKSTRUCT-
                    Struktur                          */
```

Beschreibung Die Funktion *MouseProc* ist eine bibliotheksdefinierte Callback-Funktion. Sie wird vom System immer dann aufgerufen, wenn die Anwendung die Funktion *GetMessage* oder *PeekMessage* aufruft und eine Mausbotschaft zur Bearbeitung vorliegt.

Parameter *code:* Bestimmt, ob die Callback-Funktion die Botschaft bearbeiten oder die Funktion *CallNextHookEx* aufrufen soll. Ist dieser Wert kleiner als Null, sollte die Callback-Funktion die Botschaft ohne weitere Bearbeitung an *CallNextHookEx* übergeben. Ist der Wert *HC_NOREMOVE*, hat die Anwendung die Funktion *PeekMessage* mit der Option *PM_NOREMOVE* aufgerufen. In diesem Fall wird die Botschaft nicht aus der Warteschlange des Systems entfernt.

wParam: Bestimmt den Bezeichner der Mausbotschaft.

lParam: Zeigt auf eine *MOUSEHOOKSTRUCT*-Struktur, die Informationen über die Maus enthält.

Die *MOUSEHOOKSTRUCT*-Struktur hat folgende Form:

```
typedef struct tagMOUSEHOOKSTRUCT {   /* ms */
   POINT pt;
   HWND hwnd;
   UINT wHitTestCode;
   DWORD dwExtraInfo;
} MOUSEHOOKSTRUCT;
```

Wenn die Callback-Funktion 1 zurückgibt, wird die Botschaft verworfen. Der Rückgabewert 0 erlaubt dem System die Bearbeitung der Botschaft.

Hinweis Diese Callback-Funktion sollte keine *JournalPlaybackProc*-Callback-Funktion installieren.

Anwendungen müssen diese Callback-Funktion installieren, indem sie den Filtertyp *WH_MOUSE* und die Adresse der Prozedurinstanz der Callback-Funktion in einem Aufruf der Funktion *SetWindowsHookEx* angeben.

MouseProc ist ein Platzhalter für den bibliotheksdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Bibliothek exportiert werden.

Siehe auch *CallNextHookEx, GetMessage, PeekMessage, SetWindowsHookEx*

MoveToEx 3.1

Syntax

```
BOOL MoveToEx(hdc, nX, nY, lpPoint)

HDC hdc;           /* Handle des Gerätekontexts       */
int nX;            /* x-Koordinate der neuen Position */
int nY;            /* y-Koordinate der neuen Position */
POINT FAR* lpPoint;/* Zeiger auf die Struktur mit der
                      vorherigen Position             */
```

Beschreibung Die Funktion *MoveToEx* verlagert die aktuelle Position an den Punkt, der durch die Parameter *nX* und *nY* angegeben ist. Optional wird die vorherige Position zurückgegeben.

Parameter *hdc:* Bezeichnet den Gerätekontext.

nX: Bestimmt die virtuelle x-Koordinate der neuen Position.

nY: Bestimmt die virtuelle y-Koordinate der neuen Position.

lpPoint: Zeigt auf eine *POINT*-Struktur, in der die vorherige aktuelle Position gespeichert wird. Wenn dieser Parameter *NULL* ist, wird die vorhergehende Position nicht zurückgegeben.

Die *POINT*-Struktur hat folgende Form:

```
typedef struct tagPOINT {   /* pt */
   int x;
   int y;
} POINT;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Siehe auch *MoveTo*

NotifyProc 2.x

Syntax

```
BOOL CALLBACK NotifyProc(hglbl)

HGLOBAL hglbl; /* Handle des globalen Speicherobjekts */
```

Beschreibung Die Funktion *NotifyProc* ist eine bibliotheksdefinierte Callback-Funktion, die vom System aufgerufen wird, wenn ein globales Speicherobjekt verworfen werden soll, das mit *GMEM_NOTIFY* reserviert wurde.

Parameter *hglbl:* Bezeichnet das globale Speicherobjekt, das verworfen werden soll.

Rückgabewert Die Callback-Funktion sollte einen Wert ungleich Null zurückgeben, wenn das System das Speicherobjekt verwerfen soll. Andernfalls sollte sie Null zurückgeben.

Hinweis Die Callback-Funktion muß nicht notwendigerweise im Kontext der Anwendung aufgerufen werden, der die Routine zugeordnet ist. Aus diesem Grund sollte die Callback-Funktion nicht davon ausgehen, daß sie mit dem Stack-Segment dieser Anwendung arbeitet. Die Callback-Funktion sollte keine Routine aufrufen, die Speicher verschieben könnte.

Die Callback-Funktion muß sich in einem festen Code-Segment einer dynamischen Linkbibliothek befinden.

NotifyProc ist ein Platzhalter für den bibliotheksdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Bibliothek exportiert werden.

Siehe auch *GlobalNotify*

NotifyRegister 3.1

Syntax

```
#include<toolhelp.h>
BOOL NotifyRegister(htask, lpfnCallback, wFlags)

HTASK htask;                    /* Handle der Task   */
LPFNNOTIFYCALLBACK lpfnCallback; /* Adresse der
                                    Callback-Funktion */
WORD wFlags                     /* Benachrichtigungs-
                                    Flags             */
```

Beschreibung Die Funktion *NotifyRegister* installiert eine Benachrichtigungs-Callback-Funktion für die angegebene Task.

Parameter *htask:* Bezeichnet die Task, der die Callback-Funktion zugeordnet ist. Wenn dieser Parameter *NULL* ist, bezeichnet dieser Parameter die aktuelle Task.

lpfnCallback: Zeigt auf die Benachrichtigungs-Callback-Funktion, die für die Task installiert wird. Der Kernel ruft diese Funktion immer dann auf, wenn er eine Benachrichtigung an die Task übergibt. Die Adresse einer Callback-Funktion ist normalerweise der Rückgabewert eines Aufrufs der Funktion *MakeProcInstance*. Deshalb ist beim Aufruf der Callback-Funktion das AX-Register auf den Selektor des Datensegments der

Anwendung gesetzt. Der Prolog einer exportierten Funktion enthält normalerweise folgenden Befehl:

```
mov  ds,ax
```

wFlags: Bestimmt die optionalen Benachrichtigungen, die die Anwendung zusätzlich zu den voreingestellten Benachrichtigungen erhält.

Dieser Parameter kann *NF_NORMAL* oder jede Kombination der folgenden Werte sein:

Wert	Bedeutung
NF_NORMAL	Die Anwendung erhält die voreingestellten Benachrichtigngen, aber keine der Benachrichtigungen über Task-Umschaltungen, Debug-Fehler des Systems oder Debug-Strings.
NF_TASKSWITCH	Die Anwendung erhält Task-Umschaltungsbenachrichtigungen. Dieses Flag sollte nur in Ausnahmefällen verwendet werden, da es die Geschwindigkeit des Systems stark herabsetzt.
NF_RIP	Die Anwendung erhält Benachrichtigungen über Debug-Fehler des Systems.

Rückgabewert

Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Callback-Funktion

Die Syntax der durch *lpfnCallback* bezeichneten Funktion sieht so aus:

```
BOOL NotifyRegisterCallback(wID, dwData)
WORD wID;          /* Benachrichtigungstyp          */
DWORD dwData;      /* Daten                         */
```

Parameter

wID: Bestimmt den Benachrichtigungstyp und den Wert des Parameters *dwData*.

Der Parameter *wID* kann einen der folgenden Werte haben, wenn Windows 3.0 oder eine höhere Version benutzt wird:

Wert	Bedeutung
NFY_DELMODULE	Das niederwertige *WORD* von *dwData* ist das Handle des freizugebenden Moduls.
NFY_EXITTASK	Das niederwertige Byte von *dwData* enthält den Exit-Code des Programms.
NFY_FREESEG	Das niederwertige *WORD* von *dwData* ist der Selektor des freizugebenden Segments.

Wert	Bedeutung
NFY_INCHAR	Der Parameter *dwData* wird nicht benutzt. Die Benachrichtigungs-Callback-Funktion sollte entweder den ASCII-Wert für den Tastendruck oder *NULL* zurückgeben.
NFY_LOADSEG	Der Parameter *dwData* zeigt auf eine *NFY-LOADSEG*-Strktur.
NFY_OUTSTR	Der Parameter *dwData* zeigt auf den String, der dargestellt werden soll.
NFY_RIP	Der Parameter *dwData* zeigt auf eine *NFY-RIP*-Struktur.
NFY_STARTDLL	Der Parameter *dwData* zeigt auf eine *NFY-STARTDLL*-Struktur.
NFY_STARTTASK	Der Parameter *dwData* ist der Registerinhalt von CS:IP der Startadresse der Task.
NFY_UNKNOWN	Der Kernel hat eine unbekannte Benachrichtigung zurückgegeben.

Unter Windows 3.1 kann *wID* einen der folgenden Werte annehmen:

Wert	Bedeutung
NFY_LOGERROR	Der Parameter *dwData* zeigt auf eine *NFY-LOGERROR*-Struktur.
NFY_LOGPARAM-ERROR	Der Parameter *dwData* zeigt auf eine *NFY-LOGPARAMERROR*-Struktur.
NFY_TASKIN	Der Parameter *dwData* ist nicht definiert. Die Callback-Funktion sollte die Funktion *GetCurrentTask* aufrufen.
NFY_TASKOUT	Der Parameter *dwData* ist nicht definiert. Die Callback-Funktion sollte *GetCurrentTask* aufrufen.

dwData: Bestimmt Daten, einen Zeiger auf Daten oder ist nicht definiert. Dies hängt vom Wert von *wID* ab.

Rückgabewert Der Rückgabewert der Callback-Funktion ist ungleich Null, wenn sie die Benachrichtigung bearbeitet hat. Andernfalls ist der Rückgabewert Null und die Benachrichtigung wird an andere Callback-Funktionen weitergegeben.

Hinweis Eine Benachrichtigungs-Callback-Funktion muß in der Lage sein, unbekannte Benachrichtigungswerte zu ignorieren. Benachrichtigungs-Callback-Funktionen können keine Win-

dows-Funktionen benutzen, mit Ausnahme von ToolHelp-Bibliotheksfunktionen und *PostMessage*.

NotifyRegisterCallback ist ein Platzhalter für den anwendungsdefinierten Funktionsnamen. Der tatsächliche Funktionsname muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Anwendung exportiert werden.

Siehe auch *InterruptRegister, InterruptUnRegister, MakeProcInstance, NotifyUnRegister, TerminateApp*

NotifyUnRegister 3.1

Syntax

```
#include <toolhelp.h>
BOOL NotifyUnRegister(htask)

HTASK htask;              /* Handle der Task          */
```

Beschreibung Die Funktion *NotifyUnRegister* stellt den Standard-Handler für Benachrichtigungen wieder her.

Parameter *htask:* Bezeichnet die Task. Wenn *htask NULL* ist, bezeichnet er die aktuelle Task.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Nachdem diese Funktion ausgeführt wurde, erhält die angegebene Task keine weiteren Benachrichtigungen vom Kernel.

Siehe auch *InterruptRegister, InterruptUnRegister, Notify Register, TerminateApp*

OffsetViewportOrgEx 3.1

Syntax

```
BOOL OffsetViewportOrgEx(hdc, nX, nY, lpPoint)

HDC hdc;            /* Handle des Gerätekontexts     */
int nX;             /* Geräteeinheiten, die zur
                       x-Koordinate addiert werden
                       sollen                        */
int nY;             /* Geräteeinheiten, die zur y-
                       Koordinate addiert werden
                       sollen                        */
POINT FAR* lpPoint; /* Adresse einer POINT-Struktur  */
```

Beschreibung Die Funktion *OffsetViewportOrgEx* verändert den Ursprung des Grafikfensters relativ zu den aktuellen Werten. Die folgenden Formeln sind zugrunde gelegt:

```
xNewVO = xOldVO + X
yNewVO = yOldVO + Y
```

Der neue Ursprung ist die Summe des aktuellen Ursprungs und der Werte von *nX* und *nY*.

Parameter *hdc:* Bezeichnet den Gerätekontext.

nX: Bestimmt die Anzahl der Geräteeinheiten, die zur x-Koordinate des aktuellen Ursprungs addiert werden sollen.

nY: Bestimmt die Anzahl der Geräteeinheiten, die zur y-Koordinate des aktuellen Ursprungs addiert werden sollen.

lpPoint: Zeigt auf eine *POINT*-Struktur. Der vorhergehende Ursprung des Grafikfensters wird in dieser Struktur in Gerätekoordinaten abgelegt. Wenn *lpPoint NULL* ist, wird der vorhergehende Ursprung des Grafikfensters nicht zurückgegeben.

Rückgabewert Ist die Funktion erfolgreich, ist der Rückgabewert ungleich Null, andernfalls Null.

OffsetWindowOrgEx 3.1

Syntax

```
BOOL OffsetWindowOrgEx(hdc, nX, nY, lpPoint)

HDC hdc;            /* Handle des Gerätekontexts      */
int nX;             /* virtuelle Einheiten, die zur
                       x-Koordinate addiert werden
                       sollen.                        */
int nY;             /* virtuelle Einheiten, die zur
                       y-Koordinate addiert werden
                       sollen.                        */
POINT FAR* lpPoint; /* Adresse einer POINT-Struktur   */
```

Beschreibung Die Funktion *OffsetWindowOrgEx* verändert den Ursprung des Grafikfensters relativ zu den aktuellen Werten. Die folgenden Formeln werden zugrunde gelegt:

```
xNewWO = xOldWO + X
yNewWO = yOldWO + Y
```

Der neue Ursprung ist die Summe des aktuellen Ursprungs und der Werte von *nX* und *nY*.

Parameter *hdc:* Bezeichnet den Gerätekontext.

nX: Bestimmt die Anzahl der virtuellen Einheiten, die zur x-Koordinate des aktuellen Ursprungs addiert werden sollen.

nY: Bestimmt die Anzahl der virtuellen Einheiten, die zur y-Koordinate des aktuellen Ursprungs addiert werden sollen.

lpPoint: Zeigt auf eine *POINT*-Struktur. Der vorhergehende Ursprung des Fensters wird in dieser Struktur in virtuellen Koordinaten abgelegt. Wenn *lpPoint NULL* ist, wird der vorhergehende Ursprung nicht zurückgegeben.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

OleActivate 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleActivate(lpObject, verb, fShow,
   fTakeFocus, hwnd, lpBounds);

LPOLEOBJECT lpObject;      /* Adresse des zu
                              aktivierenden Objekts   */
UINT verb;                 /* auszuführende Operation */
BOOL fShow;                /* Anzeige des Fensters ?  */
BOOL fTakeFocus;           /* Fokus an den Server
                              übergeben ?             */
HWND hwnd;                 /* Fenster-Handle des
                              Zieldokuments           */
const RECT FAR* lprcBound; /* eingrenzendes Rechteck
                              für Anzeige des Objekts */
```

Beschreibung Die Funktion *OleActivate* öffnet ein Objekt für eine an diesem Objekt durchzuführende Operation. Normalerweise wird das Objekt bearbeitet oder abgespielt.

Parameter *lpObject:* Zeigt auf das zu aktivierende Objekt.

verb: Bestimmt die durchzuführende Operation. Der Wert 0 entspricht dem primären Aktionstyp, der Wert 1 dem sekundären usw.

fShow: Bestimmt, ob das Fenster angezeigt werden soll. Ist dieser Wert *TRUE*, wird das Fenster angezeigt.

fTakeFocus: Gibt an, ob der Server den Fokus erhalten soll. Soll der Server den Fokus erhalten, muß *fTakeFocus* den Wert *TRUE* haben. Dieser Parameter ist nur von Bedeutung, wenn der Parameter *fshow* auf *TRUE* gesetzt wird.

hwnd: Bezeichnet das Fenster des Dokuments, welches das Objekt enthält. Dieser Parameter kann auf *NULL* gesetzt werden.

lpBounds: Zeigt auf eine *RECT*-Struktur, die die Koordinaten des umgebenden Rechtecks enthält, in dem das Zieldokument das Objekt anzeigt. Dieser Zeiger kann *NULL* sein. Der Abbildungsmodus des Gerätekontexts bestimmt die Einheiten dieser Koordinaten.

Rückgabewert Der Rückgabewert ist nach erfolgreicher Ausführung der Funktion *OLE_OK*. Im Fall eines Fehlers sind folgende Werte möglich:

OLE_BUSY
OLE_ERROR_OBJECT
OLE_WAIT_FOR_RELEASE

Hinweis Ein Server wird normalerweise in einem eigenen Fenster gestartet. Das Bearbeiten des Objekts erfolgt dann asynchron. Die Client-Anwendung wird durch die Callback-Funktion über die Änderungen am Objekt benachrichtigt.

Eine Client-Anwendung kann den Parameter *fShow* auf *FALSE* setzen, wenn ein Server aktiv bleiben muß, ohne dabei auf dem Bildschirm sichtbar zu sein. In diesem Fall würde die Anwendung auch die Funktion *OleSetData* benutzen.

Die Client-Anwendung bestimmt normalerweise den primären Aktionstyp, sobald der Benutzer dieses Objekt doppelt anklickt. Der Server kann jede Operation entsprechend des angegebenen Aktionstyps ausführen. Falls der Server nur eine einzige Operation unterstützt, so wird diese ausgeführt, unabhängig vom Wert des übergebenen Parameters *verb*.

Die Parameter *hwnd* und *lpBounds* können in zukünftigen Versionen des OLE-Protokolls benutzt werden, um die Position des Editierfensters des Servers zu bestimmen.

Siehe auch *OleQueryOpen, OleSetData*

OleBlockServer 3.1

Syntax
```
#include <ole.h>
OLESTATUS OleBlockServer(lhSrvr);

LHSERVER lhSrvr;     /* Handle des Server          */
```

Beschreibung Die Funktion *OleBlockServer* stellt Anfragen an den Server in eine Warteschlange, bis dieser die Funktion *OleUnblockServer* aufruft.

Parameter *lhSrvr:* Bezeichnet den Server, für den die Anfragen in eine Warteschlange gestellt werden sollen.

Rückgabewert Der Rückgabewert ist nach erfolgreicher Ausführung der Funktion *OLE_OK*. Im Fall eines Fehlers kann *OLE_ERROR_HANDLE* zurückgeliefert werden.

Hinweis Der Server muß die Funktion *OleUnblockServer* aufrufen, nachdem er die Funktion *OleBlockServer* aufgerufen hat.

Eine Server-Anwendung kann die Funktionen *OleBlockServer* und *OleUnblockServer* verwenden, um die Bearbeitung von Anfragen des Client durch die Bibliothek des Server zu steuern. Da nur die Botschaften des Client an den Server von dieser Funktion betroffen sind, kann der Server weiterhin Botschaften an die Client-Anwendung übergeben.

Einer Server-Anwendung wird beim Aufruf der Funktion *OleRegisterServer* ein Handle zugeordnet.

Siehe auch *OleRegisterServer, OleUnblockServer*

OleClone 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleClone(lpObject, lpClient, lhClientDoc,
    lpszObjname, lplpObject);

LPOLEOBJECT lpObject;        /* Adresse des zu
                                kopierenden Objekts   */
LPOLECLIENT lpClient;        /* Adresse der OLECLIENT-
                                Struktur für das neue
                                Objekt                */
LHCLIENTDOC lhClientDoc;     /* long-Handle des Client-
                                Dokuments             */
LPCSTR lpszObjname;          /* Adresse des String
                                für den Objektnamen   */
LPOLEOBJECT FAR* lplpObject; /* Adresse des Zeigers
                                auf das neue Objekt   */
```

Beschreibung Die Funktion *OleClone* erzeugt eine Kopie eines Objekts. Diese Kopie ist identisch mit dem Originalobjekt, jedoch nicht mit dem Server verbunden.

Parameter *lpObject:* Zeigt auf das zu kopierende Objekt.

lpClient: Zeigt auf eine *OLECLIENT*-Struktur für das neue Objekt.

lhClientDoc: Gibt an, in welchem Client-Dokument das Objekt erzeugt werden soll.

lpszObjname: Zeigt auf einen nullterminierten String, der den Namen der Client-Anwendung für das Objekt enthält. Dieser Name muß hinsichtlich aller anderen Namen der Objekte im Dokument eindeutig sein und darf keinen Schrägstrich (/) enthalten.

lplpObject: Zeigt auf eine Variable, in der die Bibliothek den **long**-Zeiger auf das neue Objekt speichert.

Rückgabewert Der Rückgabewert ist nach erfolgreicher Ausführung der Funktion *OLE_OK*. Im Fall eines Fehlers können folgende Fehlerwerte zurückgegeben werden:

OLE_BUSY
OLE_ERROR_HANDLE
OLE_ERROR_OBJECT
OLE_WAIT_FOR_RELEASE

Hinweis Client-Anwendungen benutzen die Funktion *OleClone* häufig, um den Befehl *Rückgängig* zu unterstützen.

Client-Anwendungen können bei Bedarf eine neue *OLECLIENT*-Struktur für das kopierte Objekt bereitstellen.

Siehe auch *OleEqual*

OleClose 3.1

Syntax
```
#include <ole.h>
OLESTATUS OleClose(lpObject);

LPOLEOBJECT lpObject;    /* Adresse des zu
                            schließenden Objekts      */
```

Beschreibung Die Funktion *OleClose* schließt das angegebene Objekt. Durch das Schließen wird die Verbindung zwischen Objekt und Server-Anwendung aufgehoben.

Parameter *lpObject:* Zeigt auf das zu schließende Objekt.

Rückgabewert Der Rückgabewert ist nach erfolgreicher Ausführung der Funktion *OLE_OK*. Im Fall eines Fehlers können folgende Fehlerwerte zurückgegeben werden:

OLE_BUSY
OLE_ERROR_HANDLE
OLE_ERROR_OBJECT
OLE_WAIT_FOR_RELEASE

Siehe auch *OleActivate, OleDelete, OleReconnect*

OleCopyFromLink 3.1

Syntax
```
#include <ole.h>
OLESTATUS OleCopyFromLink(lpObject, lpszProtocol,
   lpClient, lhClientDoc, lpszObjname, lplpObject);

LPOLEOBJECT lpObject;        /* Adresse des einzu-
                                bettenden Objekts     */
LPCSTR lpszProtocol;         /* Adresse des Protokoll-
                                namens                */
LPOLECLIENT lpClient;        /* Adresse der Client-
                                Struktur              */
LHCLIENTDOC lhClientDoc;     /* long-Handle des
                                Client-Dokuments      */
LPCSTR lpszObjname;          /* Adresse des String
                                für den Objektnamen   */
LPOLEOBJECT FAR* lplpObject; /* Adresse des Zeigers
                                auf das neue Objekt   */
```

Beschreibung Die Funktion *OleCopyFromLink* erzeugt eine eingebettete Kopie eines verknüpften Objekts.

Parameter *lpObject:* Zeigt auf das verknüpfte Objekt, das eingebettet werden soll.

lpszProtocol: Zeigt auf einen nullterminierten String, der den Namen des Protokolls angibt, das für das neue einzubettende Objekt benötigt wird. Zur Zeit kann dieser Wert StdFileEditing sein, der Name des OLE-Protokolls.

lpClient: Zeigt auf eine *OLECLIENT*-Struktur für das neue Objekt.

lhClientDoc: Bezeichnet das Client-Dokument, in dem das eingebettete Objekt erzeugt werden soll.

lpszObjname: Zeigt auf einen nullterminierten String, der den Namen des Client für das Objekt angibt.

lplpObject: Zeigt auf eine Variable, in der die Bibliothek den **long**-Zeiger auf das neue Objekt speichert.

Rückgabewert Der Rückgabewert ist nach erfolgreicher Ausführung der Funktion *OLE_OK*. Im Fall eines Fehlers kann einer der folgenden Fehlerwerte zurückgeliefert werden:

OLE_BUSY
OLE_ERROR_HANDLE
OLE_ERROR_NAME
OLE_ERROR_OBJECT
OLE_ERROR_PROTOCOL
OLE_WAIT_FOR_RELEASE

Hinweis Das Erzeugen einer eingebetteten Kopie eines verknüpften Objekts kann zum Start der Server-Anwendung führen.

Siehe auch *OleObjectConvert*

OleCopyToClipboard 3.1

Syntax
```
#include <ole.h>
OLESTATUS OleCopyToClipboard(lpObject);

LPOLEOBJECT lpObject;    /* Adresse des Objekts       */
```

Beschreibung Die Funktion *OleCopyToClipboard* kopiert das angegebene Objekt in die Zwischenablage.

Parameter *lpObject:* Zeigt auf das Objekt, das in die Zwischenablage kopiert werden soll.

Rückgabewert Der Rückgabewert ist nach erfolgreicher Ausführung der Funktion *OLE_OK*. Im Fall eines Fehlers kann *OLE_ERROR_OBJECT* zurückgegeben werden.

Hinweis

Die Funktion *OleCopyToClipboard* wird normalerweise von einer Client-Anwendung aufgerufen, wenn der Benutzer im Menü *Bearbeiten* den Befehl *Kopieren* oder *Ausschneiden* wählt.

Die Client-Anwendung sollte die Zwischenablage öffnen und leeren, die Funktion *OleCopyToClipboard* aufrufen und anschließend die Zwischenablage wieder schließen.

OleCreate 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleCreate(lpszProtocol, lpClient, lpszClass,
   lhClientDoc, lpszObjname, lplpObject, renderopt,
   cfFormat);

LPCSTR lpszProtocol;         /* Adresse des String für
                                den Protokollnamen    */
LPOLECLIENT lpClient;        /* Adresse der Client-
                                Struktur              */
LPCSTR lpszClass;            /* Adresse des String für
                                den Klassennamen      */
LHCLIENTDOC lhClientDoc;     /* long-Handle des
                                Client-Dokuments      */
LPCSTR lpszObjname;          /* Adresse des String
                                für den Objektnamen   */
LPOLEOBJECT FAR* lplpObject; /* Adresse des Zeigers
                                auf das neue Objekt   */
OLEOPT_RENDER renderopt;     /* übergabeoptionen      */
OLECLIPFORMAT cfFormat;      /* Zwischenablageformat  */
```

Beschreibung

Die Funktion *OleCreate* erzeugt ein eingebettetes Objekt der angegebenen Klasse. Der Server wird geöffnet, damit das Objekt bearbeitet werden kann.

Parameter

lpszProtocol: Zeigt auf einen nullterminierten String, der den Namen des Protokolls angibt, das für das neue einzubettende Objekt benötigt wird. Zur Zeit kann dieser Wert StdFileEditing sein, der Name des OLE-Protokolls.

lpClient: Zeigt auf eine *OLECLIENT*-Struktur für das neue Objekt.

lpszClass: Zeigt auf einen nullterminierten String, der den registrierten Namen der Klasse des zu erzeugenden Objekts bestimmt.

lhClientDoc: Bezeichnet das Client-Dokument, in dem das Objekt erzeugt werden soll.

lpszObjname: Zeigt auf einen nullterminierten String, der den Namen der Client-Anwendung für das Objekt enthält. Dieser Name muß hinsichtlich aller anderen Namen der Objekte im

Dokument eindeutig sein und darf keinen Schrägstrich (/) enthalten.

lplpObject: Zeigt auf eine Variable, in der die Bibliothek den **long**-Zeiger auf das neue Objekt ablegt.

renderopt: Bestimmt die Anforderungen des Client an die Präsentationsdaten des Objekts. Dieser Parameter kann folgende Werte annehmen:

Wert	Bedeutung
olerender_draw	Die Client-Anwendung ruft die Funktion *OleDraw* auf. Die Bibliothek erhält und verwaltet die Präsentationsdaten des Objekts.
olerender_format	Die Client-Anwendung ruft die Funktion *OleGetData* auf, um die Daten in einem bestimmten Format abzurufen. Die Bibliothek erhält und verwaltet die Daten im angeforderten Format, das durch den Parameter *cfFormat* bestimmt wird.
olerender_none	Die Client-Bibliothek erhält keine Präsentationsdaten und stellt das Objekt nicht dar.

cfFormat: Bestimmt das Format der Zwischenablage, wenn der Parameter *renderopt* den Wert *olerender_format* hat. Das Zwischenablageformat wird in einem nachfolgenden Aufruf der Funktion *OleGetData* benutzt. Ist das Zwischenablageformat *CF_METAFILEPICT*, *CF_DIB* oder *CF_BITMAP*, verwaltet die Bibliothek die Daten und stellt das Objekt dar. Die Bibliothek unterstützt keine anderen Formate bei der Darstellung.

Rückgabewert

Der Rückgabewert ist nach erfolgreicher Ausführung der Funktion *OLE_OK*. Im Fall eines Fehlers kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_HANDLE
OLE_ERROR_NAME
OLE_ERROR_PROTOCOL
OLE_WAIT_FOR_RELEASE

Hinweis

Die Übergabeoption *olerender_none* wird normalerweise zur Unterstützung hypertextartiger Verbindungen benutzt. Mit dieser Option ruft der Client nicht *OleDraw* und die Funktion *OleGetData* nur für ObjectLink-, OwnerLink- und Native-Formate auf.

Die Übergabeoption *olerender_format* erlaubt es der Client-Anwendung, Daten zu berechnen statt sie zu zeichnen. Sie kann ungewöhnliche Datenformate benutzen oder Standarddatenformate modifizieren. Ist diese Option angegeben, ruft

der Client die Funktion *OleDraw* nicht auf. Stattdessen wird *OleGetData* aufgerufen, um die Daten im angegebenen Format abzurufen.

Die Übergabeoption *olerender_draw* wird am häufigsten verwendet. Sie ist für die Client-Anwendung am einfachsten zu implementieren und bietet die größte Flexibilität. Der Client muß nur *OleDraw* aufrufen. Ein Objekt-Handler kann diese Flexibilität nutzen, indem er Präsentationsdaten nicht speichert, ein eigenes Format für diese Daten benutzt oder aus verschiedenen Formaten auswählt. Zukünftige Implementierungen von OLE werden die in dieser Option angelegte Flexibilität weitergehend ausnutzen.

Siehe auch *OleCreateFromClip, OleCreateFromTemplate, OleDraw, OleGetData*

OleCreateFromClip 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleCreateFromClip(lpszProtocol, lpClient,
   lhClientDoc, lpszObjname, lplpObject, renderopt,
   cfFormat);

LPCSTR lpszProtocol;         /* Adresse des String für
                                den Protokollnamen    */
LPOLECLIENT lpClient;        /* Adresse der Client-
                                Struktur              */
LHCLIENTDOC lhClientDoc;     /* long-Handle des
                                Client-Dokuments      */
LPCSTR lpszObjname;          /* Adresse des String für
                                den Objektnamen       */
LPOLEOBJECT FAR* lplpObject; /* Adresse des Zeigers
                                auf das neue Objekt   */
OLEOPT_RENDER renderopt;     /* übergabeoptionen      */
OLECLIPFORMAT cfFormat;      /* Zwischenablageformat  */
```

Beschreibung Die Funktion *OleCreateFromClip* erzeugt ein Objekt aus dem Inhalt der Zwischenablage.

Parameter *lpszProtocol:* Zeigt auf einen nullterminierten String, der den Namen des Protokolls angibt, das für das neue einzubettende Objekt benötigt wird. Zur Zeit kann dieser Wert StdFileEditing sein, der Name des OLE-Protokolls oder Static für nicht bearbeitbare Bilder.

lpClient: Zeigt auf eine *OLECLIENT*-Struktur, die von der Client-Anwendung zugewiesen und initialisiert wird. Der Zeiger wird zum Auffinden der Callback-Funktion benötigt und in Callback-Benachrichtigungen übergeben.

lhClientDoc: Bezeichnet das Client-Dokument, in dem das Objekt erzeugt werden soll.

lpszObjname: Zeigt auf einen nullterminierten String, der den Namen der Client-Anwendung für das Objekt enthält. Dieser Name muß hinsichtlich aller anderen Namen der Objekte im Dokument eindeutig sein und darf keinen Schrägstrich (/) enthalten.

lplpObject: Zeigt auf eine Variable, in der die Bibliothek den **long**-Zeiger auf das neue Objekt ablegt.

renderopt: Bestimmt die Anforderungen des Client an die Präsentationsdaten des Objekts. Dieser Parameter kann folgende Werte annehmen:

Wert	Bedeutung
olerender_draw	Die Client-Anwendung ruft die Funktion *OleDraw* auf. Die Bibliothek erhält und verwaltet die Präsentationsdaten des Objekts.
olerender_format	Die Client-Anwendung ruft die Funktion *OleGetData* auf, um die Daten in einem bestimmten Format abzurufen. Die Bibliothek erhält und verwaltet die Daten im angeforderten Format, das durch den Parameter *cfFormat* bestimmt wird.
olerender_none	Die Client-Bibliothek erhält keine Präsentationsdaten und stellt das Objekt nicht dar.

cfFormat: Bestimmt das Format der Zwischenablage, wenn der Parameter *renderopt* den Wert *olerender_format* hat. Das Zwischenablageformat wird in einem nachfolgenden Aufruf der Funktion *OleGetData* benutzt. Ist das Zwischenablageformat *CF_METAFILEPICT*, *CF_DIB* oder *CF_BITMAP*, verwaltet die Bibliothek die Daten und stellt das Objekt dar. Die Bibliothek unterstützt keine anderen Formate bei der Darstellung.

Rückgabewert Der Rückgabewert ist nach erfolgreicher Ausführung der Funktion *OLE_OK*.

Im Fall eines Fehlers kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_CLIP
OLE_ERROR_FORMAT
OLE_ERROR_HANDLE
OLE_ERROR_NAME
OLE_ERROR_OPTION
OLE_ERROR_PROTOCOL
OLE_WAIT_FOR_RELEASE

Hinweis

Die Client-Anwendung sollte die Zwischenablage öffnen und leeren, die Funktion *OleCreateFromClip* aufrufen und anschließend die Zwischenablage wieder schließen.

Die Übergabeoption *olerender_none* wird normalerweise zur Unterstützung hypertextartiger Verbindungen benutzt. Mit dieser Option ruft der Client nicht *OleDraw* und die Funktion *OleGetData* nur für ObjectLink-, OwnerLink- und Native-Formate auf.

Die Übergabeoption *olerender_format* erlaubt es der Client-Anwendung, Daten zu berechnen statt sie zu zeichnen. Sie kann ungewöhnliche Datenformate benutzen oder Standarddatenformate modifizieren. Ist diese Option angegeben, ruft der Client die Funktion *OleDraw* nicht auf. Stattdessen wird *OleGetData* aufgerufen, um die Daten im angegebenen Format abzurufen.

Die Übergabeoption *olerender_draw* wird am häufigsten verwendet. Sie ist für die Client-Anwendung am einfachsten zu implementieren und bietet die größte Flexibilität. Der Client muß nur *OleDraw* aufrufen. Ein Objekt-Handler kann diese Flexibilität nutzen, indem er Präsentationsdaten nicht speichert, ein eigenes Format für diese Daten benutzt oder aus verschiedenen Formaten auswählt. Zukünftige Implementierungen von OLE werden die in dieser Option angelegte Flexibilität weitergehend ausnutzen.

Siehe auch

OleCreate, OleCreateFromTemplate, OleDraw, OleGetData, OleQueryCreateFromClip

OleCreateFromFile 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleCreateFromFile(lpszProtocol, lpClient,
   lpszClass, lpszFile, lhClientDoc, lpszObjname,
   lplpObject, renderopt, cfFormat);

LPCSTR lpszProtocol;         /* Adresse des String für
                                den Protokollnamen    */
LPOLECLIENT lpClient;        /* Adresse der Client-
                                Struktur              */
LPCSTR lpszClass;            /* Adresse des String
                                für den Klassennamen  */
LPCSTR lpszFile;             /* Adresse des String
                                für den Dateinamen    */
LHCLIENTDOC lhClientDoc;     /* long-Handle des
                                Client-Dokuments      */
LPCSTR lpszObjname;          /* Adresse des String
                                für den Objektnamen   */
LPOLEOBJECT FAR* lplpObject; /* Adresse des Zeigers
                                auf das neue Objekt   */
```

```
OLEOPT_RENDER renderopt;     /* übergabeoptionen      */
OLECLIPFORMAT cfFormat;      /* Zwischenablageformat  */
```

Beschreibung Die Funktion *OleCreateFromFile* erzeugt ein eingebettetes Objekt aus dem Inhalt der angegebenen Datei.

Parameter *lpszProtocol:* Zeigt auf einen nullterminierten String, der den Namen des Protokolls angibt, das für das neue einzubettende Objekt benötigt wird. Zur Zeit kann dieser Wert StdFileEditing sein, der Name des OLE-Protokolls.

lpClient: Zeigt auf eine *OLECLIENT*-Struktur, die von der Client-Anwendung zugewiesen und initialisiert wird. Der Zeiger wird zum Auffinden der Callback-Funktion benötigt und in Callback-Benachrichtigungen übergeben.

lpszClass: Zeigt auf einen nullterminierten String, der den Namen der Klasse des neuen Objekts angibt. Ist dieser Wert *NULL*, verwendet die Bibliothek die Namenserweiterung des durch *lpszFile* bezeichneten Dateinamens, um den Klassennamen des Objekts zu ermitteln.

lpszFile: Zeigt auf einen nullterminierten String, der den Namen der Datei angibt, die das Objekt enthält.

lhClientDoc: Bezeichnet das Client-Dokument, in dem das Objekt erzeugt werden soll.

lpszObjname: Zeigt auf einen nullterminierten String, der den Namen der Client-Anwendung für das Objekt enthält. Dieser Name muß hinsichtlich aller anderen Namen der Objekte im Dokument eindeutig sein und darf keinen Schrägstrich (/) enthalten.

lplpObject: Zeigt auf eine Variable, in der die Bibliothek den **long**-Zeiger auf das neue Objekt ablegt.

renderopt: Bestimmt die Anforderungen des Client an die Präsentationsdaten des Objekts. Dieser Parameter kann folgende Werte annehmen:

Wert	Bedeutung
olerender_draw	Die Client-Anwendung ruft die Funktion *OleDraw* auf. Die Bibliothek erhält und verwaltet die Präsentationsdaten des Objekts.
olerender_format	Die Client-Anwendung ruft die Funktion *OleGetData* auf, um die Daten in einem bestimmten Format abzurufen. Die Bibliothek erhält und verwaltet die Daten im angeforderten Format, das durch den Parameter *cfFormat* bestimmt wird.

Wert	Bedeutung
olerender_none	Die Client-Bibliothek erhält keine Präsentationsdaten und stellt das Objekt nicht dar.

cfFormat: Bestimmt das Format der Zwischenablage, wenn der Parameter *renderopt* den Wert *olerender_format* hat. Das Zwischenablageformat wird in einem nachfolgenden Aufruf der Funktion *OleGetData* benutzt. Ist das Zwischenablageformat *CF_METAFILEPICT*, *CF_DIB* oder *CF_BITMAP*, verwaltet die Bibliothek die Daten und stellt das Objekt dar. Die Bibliothek unterstützt keine anderen Formate bei der Darstellung.

Rückgabewert

Der Rückgabewert ist nach erfolgreicher Ausführung der Funktion *OLE_OK*. Im Fall eines Fehlers kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_CLASS
OLE_ERROR_HANDLE
OLE_ERROR_MEMORY
OLE_ERROR_NAME
OLE_ERROR_PROTOCOL
OLE_WAIT_FOR_RELEASE

Hinweis

Wenn eine Client-Anwendung die Funktion *OleCreateFromFile* aufruft, wird der Server gestartet, um die Native- und die Präsentationsdaten zu übergeben. Anschließend wird der Server wieder geschlossen. Ist der Server zusammen mit dem Dokument bereits geöffnet, ruft diese Funktion die Informationen einfach ab, ohne den Server zu schließen. Der Server legt dem Benutzer das Objekt nicht zur Bearbeitung vor.

Die Übergabeoption *olerender_none* wird normalerweise zur Unterstützung hypertextartiger Verbindungen benutzt. Mit dieser Option ruft der Client nicht *OleDraw* und die Funktion *OleGetData* nur für ObjectLink-, OwnerLink- und Native-Formate auf.

Die Übergabeoption *olerender_format* erlaubt es der Client-Anwendung, Daten zu berechnen statt sie zu zeichnen. Sie kann ungewöhnliche Datenformate benutzen oder Standarddatenformate modifizieren. Ist diese Option angegeben, ruft der Client die Funktion *OleDraw* nicht auf. Stattdessen wird *OleGetData* aufgerufen, um die Daten im angegebenen Format abzurufen.

Die Übergabeoption *olerender_draw* wird am häufigsten verwendet. Sie ist für die Client-Anwendung am einfachsten zu implementieren und bietet die größte Flexibilität. Der Client muß nur *OleDraw* aufrufen. Ein Objekt-Handler kann diese Flexibilität nutzen, indem er Präsentationsdaten nicht speichert, ein eigenes Format für diese Daten benutzt oder aus verschiedenen Formaten auswählt. Zukünftige Implementie-

rungen von OLE werden die in dieser Option angelegte Flexibilität weitergehend ausnutzen.

Wenn eine Client-Anwendung Dateien entgegennimmt, die vom Datei-Manager abgelegt wurden, sollte sie durch Aufruf der Funktion *OleCreateFromFile* auf die *WM_DROPFILES*-Botschaft reagieren und dabei für den Parameter *lpszClass* »Packager« angeben, um auf den Objekt-Manager zu verweisen.

Siehe auch *OleCreate, OleCreateFromTemplate, OleDraw, OleGetData*

OleCreateFromTemplate 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleCreateFromTemplate(lpszProtocol, lpClient,
   lpszTemplate, lhClientDoc, lpszObjname, lplpObject,
   renderopt, cfFormat);

LPCSTR lpszProtocol;         /* Adresse des String für
                                den Protokollnamen    */
LPOLECLIENT lpClient;        /* Adresse der Client-
                                Struktur              */
LPCSTR lpszTemplate;         /* Adresse des String mit
                                dem Pfad der Datei    */
LHCLIENTDOC lhClientDoc;     /* long-Handle des
                                Client-Dokuments      */
LPCSTR lpszObjname;          /* Adresse des String für
                                den Objektnamen       */
LPOLEOBJECT FAR* lplpObject; /* Adresse des Zeigers
                                auf das neue Objekt   */
OLEOPT_RENDER renderopt;     /* übergabeoptionen      */
OLECLIPFORMAT cfFormat;      /* Zwischenablageformat  */
```

Beschreibung Die Funktion *OleCreateFromTemplate* erzeugt ein Objekt, indem ein anderes Objekt als Schablone benutzt wird. Der Server wird geöffnet, um die anfängliche Bearbeitung des Objekts zu ermöglichen.

Parameter *lpszProtocol:* Zeigt auf einen nullterminierten String, der den Namen des Protokolls angibt, das für das neue einzubettende Objekt benötigt wird. Zur Zeit kann dieser Wert StdFileEditing sein, der Name des OLE-Protokolls.

lpClient: Zeigt auf eine *OLECLIENT*-Struktur, die von der Client-Anwendung zugewiesen und initialisiert wird.

lpszTemplate: Zeigt auf einen nullterminierten String, der den Pfad der Datei angibt, die als Schablone für das neue Objekt dienen soll. Der Server wird zur Bearbeitung geöffnet. Er lädt den anfänglichen Status des neuen Objekts aus der angegebenen Datei.

lhClientDoc: Bezeichnet das Client-Dokument, in dem das Objekt erzeugt werden soll.

lpszObjname: Zeigt auf einen nullterminierten String, der den Namen der Client-Anwendung für das Objekt enthält. Dieser Name muß hinsichtlich aller anderen Namen der Objekte im Dokument eindeutig sein und darf keinen Schrägstrich (/) enthalten.

lplpObject: Zeigt auf eine Variable, in der die Bibliothek den **long**-Zeiger auf das neue Objekt ablegt.

renderopt: Bestimmt die Anforderungen des Client an die Präsentationsdaten des Objekts. Dieser Parameter kann folgende Werte annehmen:

Wert	Bedeutung
olerender_draw	Die Client-Anwendung ruft die Funktion *OleDraw* auf. Die Bibliothek erhält und verwaltet die Präsentationsdaten des Objekts.
olerender_format	Die Client-Anwendung ruft die Funktion *OleGetData* auf, um die Daten in einem bestimmten Format abzurufen. Die Bibliothek erhält und verwaltet die Daten im angeforderten Format, das durch den Parameter *cfFormat* bestimmt wird.
olerender_none	Die Client-Bibliothek erhält keine Präsentationsdaten und stellt das Objekt nicht dar.

cfFormat: Bestimmt das Format der Zwischenablage, wenn der Parameter *renderopt* den Wert *olerender_format* hat. Das Zwischenablageformat wird in einem nachfolgenden Aufruf der Funktion *OleGetData* benutzt. Ist das Zwischenablageformat *CF_METAFILEPICT*, *CF_DIB* oder *CF_BITMAP*, verwaltet die Bibliothek die Daten und stellt das Objekt dar. Die Bibliothek unterstützt keine anderen Formate bei der Darstellung.

Rückgabewert

Der Rückgabewert ist nach erfolgreicher Ausführung der Funktion *OLE_OK*. Im Fall eines Fehlers kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_CLASS
OLE_ERROR_HANDLE
OLE_ERROR_MEMORY
OLE_ERROR_NAME
OLE_ERROR_PROTOCOL
OLE_WAIT_FOR_RELEASE

Hinweis

Die Client-Bibliothek verwendet die Namenserweiterung der im Parameter *lpszTemplate* angegebenen Datei, um den Server

für das Objekt zu ermitteln. Die Verknüpfung zwischen der Namenserweiterung und dem Server ist in der Registrierungsdatenbank gespeichert.

Die Übergabeoption *olerender_none* wird normalerweise zur Unterstützung hypertextartiger Verbindungen benutzt. Mit dieser Option ruft der Client nicht *OleDraw* und die Funktion *OleGetData* nur für ObjectLink-, OwnerLink- und Native-Formate auf.

Die Übergabeoption *olerender_format* erlaubt es der Client-Anwendung, Daten zu berechnen statt sie zu zeichnen. Sie kann ungewöhnliche Datenformate benutzen oder Standarddatenformate modifizieren. Ist diese Option angegeben, ruft der Client die Funktion *OleDraw* nicht auf. Stattdessen wird *OleGetData* aufgerufen, um die Daten im angegebenen Format abzurufen.

Die Übergabeoption *olerender_draw* wird am häufigsten verwendet. Sie ist für die Client-Anwendung am einfachsten zu implementieren und bietet die größte Flexibilität. Der Client muß nur *OleDraw* aufrufen. Ein Objekt-Handler kann diese Flexibilität nutzen, indem er Präsentationsdaten nicht speichert, ein eigenes Format für diese Daten benutzt oder aus verschiedenen Formaten auswählt. Zukünftige Implementierungen von OLE werden die in dieser Option angelegte Flexibilität weitergehend ausnutzen.

Siehe auch *OleCreate, OleCreateFromClip, OleDraw, OleGetData, OleObjectConvert*

OleCreateInvisible 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleCreateInvisible(lpszProtocol, lpClient,
    lpszClass, lhClientDoc, lpszObjname, lplpObject,
    renderopt, cfFormat, fActivate)

LPCSTR lpszProtocol;         /* Adresse des Proto-
                                kollnamen-String      */
LPOLECLIENT lpClient;        /* Adresse der Client-
                                Struktur              */
LPCSTR lpszClass;            /* Adresse des Klassen-
                                namen-String          */
LHCLIENTDOC lhClientDoc;     /* long-Handle des
                                Client-Dokuments      */
LPCSTR lpszObjname;          /* Adresse des Objekt-
                                namen-String          */
LPOLEOBJECT FAR* lplpObject; /* Adresse des Zeigers
                                auf das Objekt        */
OLEOPT_RENDER renderopt;     /* übergabeoptionen      */
OLECLIPFORMAT cfFormat;      /* Zwischenablageformat  */
```

```
BOOL fActivate;              /* Aktivierungs-Flag des
                               Server                */
```

Beschreibung Die Funktion *OleCreateInvisible* erzeugt ein Objekt, ohne daß die Server-Anwendung für den Benutzer angezeigt wird. Sie startet entweder den Server, um das Objekt zu erzeugen oder erzeugt selbst ein leeres Objekt mit der angegebenen Klasse und dem angegebenen Format.

Parameter *lpszProtocol:* Zeigt auf einen nullterminierten String, der den Namen des Protokolls enthält, das für das neue eingebettete Objekt benötigt wird. Zur Zeit kann dieser Wert StdFileEditing (der Name des OLE-Protokolls) oder Static (für nicht bearbeitbare Bilder) sein.

lpClient: Zeigt auf eine von der Client-Anwendung reservierte und initialisierte *OLECLIENT*-Struktur. Dieser Zeiger wird zum Auffinden der richtigen Callback-Funktion verwendet und in Callback-Benachrichtigungen übergeben.

lpszClass: Zeigt auf einen nullterminierten String, der den registrierten Klassennamen des Objekts enthält, das erzeugt werden soll.

lhClientDoc: Bezeichnet das Client-Dokument, in dem das Objekt erzeugt wird.

lpszObjname: Zeigt auf einen nullterminierten String, der den Namen des Client für das Objekt angibt. Dieser Name muß in Bezug auf die anderen Objektnamen im Dokument eindeutig sein und darf keine Schrägstriche (/) enthalten.

lplpObject: Zeigt auf eine Variable, in der die Bibliothek den **long**-Zeiger auf das neue Objekt speichert.

renderopt: Bestimmt die Vorgabe des Client für Präsentationsdaten für das Objekt. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
olerender_draw	Die Client-Anwendung ruft die Funktion *OleDraw* auf. Die Bibliothek fordert die Präsentationsdaten an und verwaltet sie.
olerender_format	Die Client-Anwendung ruft die Funktion *OleGetData* auf, um Daten in einem bestimmten Format abzurufen. Die Bibliothek fordert die Daten in dem in *cfFormat* angegebenen Format an und verwaltet sie.
olerender_none	Die Client-Bibliothek fordert keine Präsentationsdaten an und zeichnet das Objekt nicht.

cfFormat: Bestimmt das Zwischenablageformat, wenn der Parameter *renderopt* den Wert *olerender_format* hat. Dieses Format wird bei einem nachfolgenden Aufruf der Funktion *OleGetData* übergeben. Wenn es den Wert *CF_METAFILEPICT*, *CF_DIB* oder *CF_BITMAP* hat, verwaltet die Bibliothek die Daten und zeichnet das Objekt. Bei Angabe eines anderen Formats wird das Objekt nicht gezeichnet.

fActivate: Bestimmt, ob der Server für das Objekt gestartet wird. Wenn dieser Parameter *TRUE* ist, wird die Server-Anwendung gestartet, aber nicht angezeigt. Wenn er *FALSE* ist, wird die Server-Anwendung nicht gestartet und die Funktion erzeugt ein leeres Objekt mit der angegebenen Klasse und dem angegebenen Format.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_HANDLE
OLE_ERROR_NAME
OLE_ERROR_PROTOCOL

Hinweis Eine Anwendung kann verhindern, daß ein Objekt wiederholt neu gezeichnet wird. Sie muß dazu die Funktion *OleCreateInvisible* aufrufen, bevor sie das Objekt mit Funktionen wie *OleSetBounds*, *OleSetColorScheme* und *OleSetTargetDevice* initialisiert. Danach kann sie das Objekt entweder mit Hilfe der Funktion *OleActivate* anzeigen oder es durch *OleUpdate* und *OleClose* aktualisieren, ohne es anzuzeigen.

Siehe auch *OleActivate*, *OleClose*, *OleSetBounds*, *OleSetColorScheme*, *OleSetTargetDevice*, *OleUpdate*

OleCreateLinkFromClip 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleCreateLinkFromClip(lpszProtocol, lpClient,
   lhClientDoc, lpszObjname, lplpObject, renderopt,
   cfFormat);

LPCSTR lpszProtocol;         /* Adresse des String für
                                den Protokollnamen    */
LPOLECLIENT lpClient;        /* Adresse der Client-
                                Struktur              */
LHCLIENTDOC lhClientDoc;     /* long-Handle des
                                Client-Dokuments      */
LPCSTR lpszObjname;          /* Adresse des String
                                für den Objektnamen   */
LPOLEOBJECT FAR* lplpObject; /* Adresse des Zeigers
                                auf das neue Objekt   */
```

```
OLEOPT_RENDER renderopt;     /* übergabeoptionen      */
OLECLIPFORMAT cfFormat;      /* Zwischenablageformat  */
```

Beschreibung Die Funktion *OleCreateLinkFromClip* erzeugt eine Verknüpfung mit einem Objekt in der Zwischenablage.

Parameter *lpszProtocol:* Zeigt auf einen nullterminierten String, der den Namen des Protokolls angibt, das für das neue einzubettende Objekt benötigt wird. Zur Zeit kann dieser Wert StdFileEditing sein, der Name des OLE-Protokolls.

lpClient: Zeigt auf eine *OLECLIENT*-Struktur, die von der Client-Anwendung zugewiesen und initialisiert wird. Dieser Zeiger wird zum Auffinden der Callback-Funktion verwendet und in Callback-Benachrichtigungen übergeben.

lhClientDoc: Bezeichnet das Client-Dokument, in dem das Objekt erzeugt werden soll.

lpszObjname: Zeigt auf einen nullterminierten String, der den Namen der Client-Anwendung für das Objekt enthält. Dieser Name muß hinsichtlich aller anderen Namen der Objekte im Dokument eindeutig sein und darf keinen Schrägstrich (/) enthalten.

lplpObject: Zeigt auf eine Variable, in der die Bibliothek den **long**-Zeiger auf das neue Objekt ablegt.

renderopt: Bestimmt die Anforderungen des Client an die Präsentationsdaten des Objekts. Dieser Parameter kann folgende Werte annehmen:

Wert	Bedeutung
olerender_draw	Die Client-Anwendung ruft die Funktion *OleDraw* auf. Die Bibliothek erhält und verwaltet die Präsentationsdaten des Objekts.
olerender_format	Die Client-Anwendung ruft die Funktion *OleGetData* auf, um die Daten in einem bestimmten Format abzurufen. Die Bibliothek erhält und verwaltet die Daten im angeforderten Format, das durch den Parameter *cfFormat* bestimmt wird.
olerender_none	Die Client-Bibliothek erhält keine Präsentationsdaten und stellt das Objekt nicht dar.

cfFormat: Bestimmt das Format der Zwischenablage, wenn der Parameter *renderopt* den Wert *olerender_format* hat. Das Zwischenablageformat wird in einem nachfolgenden Aufruf der Funktion *OleGetData* benutzt. Ist das Zwischenablageformat *CF_METAFILEPICT*, *CF_DIB* oder *CF_BITMAP*, verwaltet die

Bibliothek die Daten und stellt das Objekt dar. Die Bibliothek unterstützt keine anderen Formate bei der Darstellung.

Rückgabewert Der Rückgabewert ist nach erfolgreicher Ausführung der Funktion *OLE_OK*. Im Fall eines Fehlers kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_CLIP
OLE_ERROR_FORMAT
OLE_ERROR_HANDLE
OLE_ERROR_NAME
OLE_ERROR_PROTOCOL
OLE_WAIT_FOR_RELEASE

Hinweis Die Übergabeoption *olerender_none* wird normalerweise zur Unterstützung hypertextartiger Verbindungen benutzt. Mit dieser Option ruft der Client nicht *OleDraw* und die Funktion *OleGetData* nur für ObjectLink-, OwnerLink- und Native-Formate auf.

Die Übergabeoption *olerender_format* erlaubt es der Client-Anwendung, Daten zu berechnen statt sie zu zeichnen. Sie kann ungewöhnliche Datenformate benutzen oder Standarddatenformate modifizieren. Ist diese Option angegeben, ruft der Client die Funktion *OleDraw* nicht auf. Stattdessen wird *OleGetData* aufgerufen, um die Daten im angegebenen Format abzurufen.

Die Übergabeoption *olerender_draw* wird am häufigsten verwendet. Sie ist für die Client-Anwendung am einfachsten zu implementieren und bietet die größte Flexibilität. Der Client muß nur *OleDraw* aufrufen. Ein Objekt-Handler kann diese Flexibilität nutzen, indem er Präsentationsdaten nicht speichert, ein eigenes Format für diese Daten benutzt oder aus verschiedenen Formaten auswählt. Zukünftige Implementierungen von OLE werden die in dieser Option angelegte Flexibilität weitergehend ausnutzen.

Siehe auch *OleCreate, OleCreateFromTemplate, OleDraw, OleGetData, OleQueryLinkFromClip*

OleCreateLinkFromFile 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleCreateLinkFromFile(lpszProtocol, lpClient,
   lpszClass, lpszFile, lpszItem, lhClientDoc,
   lpszObjname, lplpObject, renderopt, cfFormat)

LPCSTR lpszProtocol;         /* Adresse des String für
                                den Protokollnamen    */
LPOLECLIENT lpClient;        /* Adresse der Client-
                                Struktur              */
```

```
LPCSTR lpszClass;            /* String für den
                                Klassennamen          */
LPCSTR lpszFile;             /* Adresse des String
                                für den Dateinamen    */
LPCSTR lpszItem;             /* Adresse des String für
                                den zu verknüpfenden
                                Teil des Dokuments    */
LHCLIENTDOC lhClientDoc;     /* long-Handle des
                                Client-Dokuments      */
LPCSTR lpszObjname;          /* Adresse des String
                                für den Objektnamen   */
LPOLEOBJECT FAR* lplpObject; /* Adresse des Zeigers
                                auf das neue Objekt   */
OLEOPT_RENDER renderopt;     /* übergabeoptionen      */
OLECLIPFORMAT cfFormat;      /* Zwischenablageformat  */
```

Beschreibung Diese Funktion erzeugt ein verknüpftes Ojekt aus einer Datei, die ein Objekt enthält. Falls notwendig, startet die Bibliothek den Server, um die Präsentationsdaten zu übergeben. Das Objekt wird aber nicht zum Bearbeiten angezeigt.

Parameter *lpszProtocol:* Zeigt auf einen nullterminierten String, der den Namen des benötigten Protokolls bestimmt. Zur Zeit kann dieser Wert StdFileEditing sein, der Name des OLE-Protokolls.

lpClient: Zeigt auf eine *OLECLIENT*-Struktur, deren Speicherplatz von der Client-Anwendung reserviert wird. Außerdem initialisiert die Client-Anwendung die Struktur. Dieser Zeiger wird verwendet, um die richtige Callback-Funktion aufzufinden. Er wird in Callback-Benachrichtigungen übergeben.

lpszClass: Zeigt auf einen nullterminierten String, der den Klassennamen für das neue Objekt enthält. Wenn dieser Wert *NULL* ist, verwendet die Bibliothek die Namenserweiterung des Dateinamens, auf den der Parameter *lpszFile* zeigt, um den Klassennamen für das Objekt zu ermitteln.

lpszFile: Zeigt auf einen nullterminierten String, der den Namen der Datei bestimmt, die das Objekt enthält.

lpszItem: Zeigt auf einen nullterminierten String, der den Teil des Dokuments bezeichnet, auf den sich die Verknüpfung beziehen soll. Wenn dieser Wert *NULL* ist, bezieht sich die Verknüpfung auf das gesamte Dokument.

lhClientDoc: Bezeichnet das Client-Dokument, in dem das Objekt erzeugt wird.

lpszObjname: Zeigt auf einen nullterminierten String, der den Client-Namen des Objekts bestimmt. Dieser Name muß im Hinblick auf die Namen aller anderen Objekte im Dokument eindeutig sein und darf keinen Schrägstrich enthalten (/).

lplpObject: Zeigt auf eine Variable, in der die Bibliothek den **long**-Zeiger auf das neue Objekt speichert.

renderopt: Bestimmt die bevorzugten Präsentationsdaten des Client für das Objekt. Dieser Parameter kann folgende Werte annehmen:

Wert	Bedeutung
olerender_draw	Der Client ruft die Funktion *OleDraw* auf. Die Bibliothek erhält und verwaltet die Präsentationsdaten.
olerender_format	Der Client ruft die Funktion *OleGetData* auf, um die Daten in einem bestimmten Format abzurufen. Die Bibliothek erhält und verwaltet die Daten im angeforderten Format, das im Parameter *cfFormat* bestimmt wurde.
olerender_none	Die Client-Bibliothek erhält keine Präsentationsdaten und zeichnet das Objekt nicht.

cfFormat: Bestimmt das Format der Zwischenablage, wenn der Parameter *renderopt* den Wert *olerender_format* hat. Dieses Format wird in einem nachfolgenden Aufruf von *OleGetData* verwendet. Ist das Zwischenablageformat *CF_METAFILEPICT*, *CF_DIB* oder *CF_BITMAP*, verwaltet die Bibliothek die Daten und zeichnet das Objekt. Die Bibliothek unterstützt das Zeichnen in keinem anderen Format.

Rückgabewert

Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_CLASS
OLE_ERROR_HANDLE
OLE_ERROR_MEMORY
OLE_ERROR_NAME
OLE_ERROR_PROTOCOL
OLE_WAIT_FOR_RELEASE

Hinweis

Die Übergabeoption *olerender_none* wird normalerweise verwendet, um hypertextartige Verbindungen zu unterstützen. Mit dieser Option ruft der Client nicht die Funktion *OleDraw* und die Funktion *OleGetData* nur für ObjectLink-, OwnerLink- und Native-Formate auf.

Die Übergabeoption *olerender_format* erlaubt einem Client, Daten zu berechnen anstatt sie zu zeichnen. Außerdem kann er ungewöhnliche Datenformate benutzen oder ein Standarddatenformat verändern. Mit dieser Option ruft der Client nicht die Funktion *OleDraw* auf. Sein Aufruf von *OleGetData* liefert die Daten im angegebenen Format zurück.

Die gebräuchlichste Übergabeoption ist *olerender_draw*. Sie ist die für den Client am leichtesten zu implementierende Übergabeoption (der Client muß nur *OleDraw* aufrufen), und sie ist sehr flexibel in der Verwendung. Ein Objekt-Handler kann diese Flexibilität ausnutzen, um entweder überhaupt keine Präsentationsdaten, ein privates Datenformat oder mehrere verschiedene Formate, zwischen denen frei gewählt werden kann, zu speichern. Zukünftige Implementierungen von OLE werden die Flexibilität dieser Option weitergehend ausnutzen.

Siehe auch *OleCreate*, *OleCrateFromFile*, *OleCreateFromTemplate*, *OleDraw*, *OleGetData*

OleDelete 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleDelete(lpObject)

LPOLEOBJECT lpObject;      /*  Adresse des zu
                               löschenden Objekts     */
```

Beschreibung Diese Funktion löscht ein Objekt und gibt den zugeordneten Speicher frei. Falls das Objekt geöffnet war, wird es geschlossen.

Parameter *lpObject:* Zeigt auf das zu löschende Objekt.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_BUSY
OLE_ERROR_OBJECT
OLE_WAIT_FOR_RELEASE

Hinweis Eine Anwendung verwendet die Funktion *OleDelete*, wenn das Objekt nicht mehr Teil eines Client-Dokuments ist.

Die Funktion *OleDelete* zeigt im Gegensatz zu *OleRelease* an, daß ein Objekt dauerhaft entfernt worden ist.

Siehe auch *OleClose*, *OleRelease*

OleDraw 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleDraw(lpObject, hdc, lprcBounds,
   lprcWBounds, hdcFormat)

LPOLEOBJECT lpObject;        /* Adresse des zu
                                zeichnenden Objekts   */
```

```
HDC hdc;                        /* Handle des Geräte-
                                   kontexts für das
                                   Objekt               */
const RECT FAR* lprcBounds;     /* umgebendes Rechteck
                                   für das Objekt       */
const RECT FAR* lprcWBounds;    /* umgebendes Rechteck
                                   für den Metafile-
                                   Gerätekontext        */
HDC hdcFormat;                  /* Handle des Gerätekon-
                                   texts für die Forma-
                                   tierung des Objekts  */
```

Beschreibung Diese Funktion zeichnet ein angegebenes Objekt in ein umgebendes Rechteck in einem Gerätekontext.

Parameter *lpObject:* Zeigt auf das zu zeichnende Objekt.

hdc: Bezeichnet den Gerätekontext, in dem das Objekt gezeichnet werden soll.

lprcBounds: Zeigt auf eine *RECT*-Struktur, die das umgebende Rechteck für den durch *hdc* bestimmten Gerätekontext festlegt, in dem das Objekt gezeichnet werden soll.

lprcWBounds: Zeigt auf eine *RECT*-Struktur, die das umgebende Rechteck festlegt, wenn der Parameter *hdc* ein Metafile bestimmt. Die Elemente *left* und *top* der *RECT*-Struktur sollten den Ursprung des Fensters, die Elemente *right* und *bottom* die Abmessungen des Fensters bestimmen.

hdcFormat: Bezeichnet einen Gerätekontext, der das Zielgerät beschreibt, für das das Objekt formatiert werden soll.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_ABORT
OLE_ERROR_BLANK
OLE_ERROR_DRAW
OLE_ERROR_MEMORY
OLE_ERROR_OBJECT

Die Funktion liefert *OLE_ERROR_ABORT* zurück, wenn die Callback-Funktion während des Zeichnens *FALSE* zurückgibt.

Wenn der Parameter *hdc* den Gerätekontext eines Metafile bestimmt, ist das durch den Parameter *lprcWBounds* bestimmte Rechteck das Rechteck, das auch der Parameter *lprcBounds* angibt. Wenn *hdc* nicht den Gerätekontext eines Metafile angibt, wird der Parameter *lprcWBounds* ignoriert.

Die Bibliothek kann einen Objekt-Handler benutzen, um das Objekt zu übergeben. Dieser Handler benötigt möglicherweise Informationen über das Zielgerät. Deshalb muß ein

Handle für einen Gerätekontext im Parameter *hdcFormat* angegeben werden. Der Parameter *lprcBounds* bezeichnet das Rechteck im Gerätekontext (relativ zum aktuellen Abbildungsmodus), auf das das Objekt abbgebildet werden soll. Dies kann auch ein Skalieren des Bildes mit sich bringen und kann von Client-Anwendungen dazu benutzt werden, eine Skalierung zwischen der angezeigten Darstellung und dem gedruckten Bild durchzuführen.

Ein Objekt-Handler sollte ein Objekt so formatieren, als ob es in der Größe gezeichnet werden sollte, die beim Aufruf der Funktion *OleSetBounds* für den durch *hdcFormat* angegebenen Gerätekontext bestimmt wird. Häufig wird dieses Formatieren bereits von der Server-Anwendung durchgeführt. In diesem Fall übernimmt die Bibliothek die Präsentationsdaten mit einer geeigneten Skalierung für das umgebende Rechteck. Wenn das Objekt verkleinert werden soll, kann der Gerätekontext für die Ausgabe eine Clipping-Region enthalten, die kleiner als das angegebene umgebende Rechteck ist.

Siehe auch *OleSetBounds*

OleEnumFormats 3.1

Syntax

```
#include <ole.h>
OLECLIPFORMAT OleEnumFormats(lpObject, cfFormat)

LPOLEOBJECT lpObject;    /* Adresse des abzufragenden
                            Objekts                  */
OLECLIPFORMAT cfFormat;  /* Format des vorherigen
                            Funktionsaufrufs         */
```

Beschreibung Diese Funktion zählt die Datenformate auf, die ein bestimmtes Objekt beschreiben.

Parameter *lpObject:* Zeigt auf das abzufragende Objekt.

cfFormat: Bestimmt das Format, das beim letzten Aufruf von *OleEnumFormats* zurückgeliefert wurde. Beim ersten Aufruf dieser Funktion sollte dieser Parameter Null sein.

Rückgabewert Der Rückgabewert ist das nächste verfügbare Format, sofern weitere Formate zur Verfügung stehen. Andernfalls ist der Rückgabewert *NULL*.

Hinweis Wenn eine Anwendung Null für den Parameter *cfFormat* angibt, liefert *OleEnumFormats* das erste verfügbare Format. Wenn eine Anwendung ein Format angibt, das ein vorhergehender Aufruf der Funktion *OleEnumFormats* zurückgeliefert hat, gibt die Funktion das folgende Format zurück. Sind keine weiteren Formate vorhanden, liefert die Funktion *NULL*.

Siehe auch *OleGetData*

OleEnumObjects 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleEnumObjects(lhDoc, lplpObject)

LHCLIENTDOC lhDoc;            /* Handle des Dokuments */
LPOLEOBJECT FAR* lplpObject;  /* Adresse des Zeigers
                                 auf das Objekt       */
```

Beschreibung Diese Funktion listet die Objekte in einem angegebenen Dokument auf.

Parameter *lhDoc:* Bezeichnet das Dokument, für das die Objekte aufgelistet werden sollen.

lplpObject: Zeigt auf ein Objekt im Dokument, wenn die Funktion zurückkehrt. Beim ersten Aufruf dieser Funktion sollte dieser Parameter auf ein *NULL*-Objekt zeigen.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_HANDLE
OLE_ERROR_OBJECT

Hinweis Wenn eine Anwendung ein *NULL*-Objekt für den Parameter *lplpObject* angibt, liefert die Funktion *OleEnumObjects* das erste Objekt im Dokument. Wenn eine Anwendung ein Objekt angibt, das ein vorhergender Aufruf der Funktion *OleEnumObjects* zurückgeliefert hat, gibt die Funktion das folgende Objekt zurück. Sind keine weiteren Objekte im Dokument vorhanden, zeigt der Parameter *lplpObject* auf ein *NULL*-Objekt.

Diese Funktion listet nur Objekte auf, die geladen und nicht freigegeben wurden.

Siehe auch *OleDelete*, *OleRelease*

OleEqual 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleEqual(lpObject1, lpObject2)

LPOLEOBJECT lpObject1;  /* Adresse des ersten Objekts
                           für den Vergleich         */
LPOLEOBJECT lpObject2;  /* Adresse des zweiten Objekts
                           für den Vergleich         */
```

Beschreibung Diese Funktion vergleicht zwei Objekte auf Identität.

Parameter *lpObject:* Zeigt auf das erste Objekt, das auf Gleichheit getestet werden soll.

lpObject: Zeigt auf das zweite Objekt, das auf Gleichheit getestet werden soll.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_OBJECT
OLE_ERROR_NOT_EQUAL

Hinweis Eingebettete Objekte sind gleich, wenn ihre Klasse, ihr Element und ihre Native-Daten gleich sind. Verknüpfte Objekte sind gleich, wenn ihre Klasse, ihr Dokument und ihr Element gleich sind.

Siehe auch *OleClone*, *OleQueryOutOfData*

OleExecute 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleExecute(lpObject, hglbCmds, reserved)

LPOLEOBJECT lpObject;  /* Adresse des Objekts, das
                          die DDE-Befehle empfängt    */
HGLOBAL hglbCmds;      /* Handle des Puffers
                          mit Befehlen                */
UINT reserved;         /* reserviert                  */
```

Beschreibung Diese Funktion sendet DDE-Befehle an den Server des angegebenen Objekts.

Parameter *lpObject:* Zeigt auf ein Objekt, das den Server bezeichnet, der die DDE-Befehle erhalten soll.

hglbCmds: Bezeichnet den Speicherpuffer, der die DDE-Befehle enthält.

reserved: Ist reserviert und muß Null sein.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_BUSY
OLE_ERROR_COMMAND
OLE_ERROR_MEMORY
OLE_ERROR_NOT_OPEN
OLE_ERROR_OBJECT
OLE_ERROR_PROTOCOL
OLE_ERROR_STATIC
OLE_WAIT_FOR_RELEASE

Hinweis Die Client-Anwendung sollte vor einem Aufruf von *OleExecute* die Funktion *OleQueryProtocol* aufrufen und dabei StdExecute

angeben. Die Funktion *OleQueryProtocol* ist erfolgreich, wenn der Server eines Objekts die Funktion *OleExecute* unterstützt.

Siehe auch *OleQueryProtocol*

OleGetData 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleGetData(lpObject, cfFormat, lphData)

LPOLEOBJECT lpObject;   /* Adresse des abzufragenden
                           Objekts                  */
OLECLIPFORMAT cfFormat; /* Format der abgerufenen
                           Daten                    */
HANDLE FAR* lphData;    /* Adresse des Speicherobjekts,
                           das die Daten aufnehmen
                           soll                     */
```

Beschreibung Diese Funktion ruft Daten im angegebenen Format vom angegebenen Objekt ab. Die Funktion stellt das Handle für das Speicherobjekt oder das GDI-Objekt bereit, das die Daten enthält.

Parameter *lpObject:* Zeigt auf das Objekt, von dem die Daten abgerufen werden sollen.

cfFormat: Bestimmt das Format, in dem die Daten zurückgeliefert werden. Dieser Parameter kann eins der vordefinierten Formate der Zwischenablage oder der Wert sein, den die Funktion *RegisterClipboardFormat* zurückgibt.

lphData: Zeigt bei Rückkehr der Funktion auf das Handle des Speicherobjekts, das die Daten enthält.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_BLANK
OLE_ERROR_FORMAT
OLE_ERROR_OBJECT
OLE_WARN_DELETE_DATA

Hinweis Wenn die Funktion *OleGetData OLE_WARN_DELETE_DATA* zurückgibt, sind die Daten der Client-Anwendung zugeordnet. Diese sollte den Speicher der Daten freigeben, wenn sie nicht mehr benötigt werden. Bei anderen Rückgabewerten sollte der Client den Speicher nicht freigeben und die Daten nicht verändern, weil die Daten in diesen Fällen von der Client-Bibliothek verwaltet werden. Benötigt eine Anwendung die Daten für einen längeren Zeitraum, sollten diese kopiert werden.

Die Funktion *OleGetData* liefert normalerweise *OLE_WARN_DELETE_DATA*, wenn ein Objekt-Handler Daten für ein Objekt erzeugt, die die Client-Bibliothek nicht interpretieren kann. In diesem Fall muß die Client-Anwendung die Daten verwalten.

Falls die Funktion *OleGetData CF_METAFILE* oder *CF_BITMAP* bestimmt, zeigt der Parameter *lphData* bei Rückkehr der Funktion auf ein GDI-Objekt und nicht auf ein Speicherobjekt. Für alle anderen Formate stellt *OleGetData* das Handle für ein Speicherobjekt bereit.

Siehe auch *OleEnumFormats*, *OleSetData*, *RegisterClipboardFormat*

OleGetLinkUpdateOptions 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleGetLinkUpdateOptions(lpObject, lpUpdateOpt)

LPOLEOBJECT lpObject;          /* Adresse des abzu-
                                  fragenden Objekts  */
OLEOPT_UPDATE FAR* lpUpdateOpt; /* Adresse der Aktua-
                                  lisierungsoptionen */
```

Beschreibung Diese Funktion liefert die Aktualisierungsoptionen für die Verknüpfung der Präsentation eines angegebenen Objekts.

Parameter *lpObject:* Zeigt auf das abzufragende Objekt.

lpUpdateOptions: Zeigt auf eine Variable, in der die Funktion den aktuellen Wert der Aktualisierungsoptionen für die Verknüpfung des angegebenen Objekts speichert.

Die Aktualisierungsoptionen der Verknüpfung können auf folgende Werte gesetzt werden:

Wert	Bedeutung
oleupdate_always	Sofortige Aktualisierung des verknüpften Objekts. Diese Option unterstützt das Schaltfeld *Aktualisieren: Automatisch* im Dialogfenster *Verknüpfungen*.
oleupdate_oncall	Das verknüpfte Objekt wird nur auf Anforderung der Client-Anwendung aktualisiert. Diese Option unterstützt das Schaltfeld *Aktualisieren: Manuell* im Dialogfenster *Verknüpfungen*.
oleupdate_onsave	Das verknüpfte Objekt wird aktualisiert, wenn das Quelldokument vom Server gespeichert wird.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_OBJECT
OLE_ERROR_STATIC

Siehe auch *OleSetLinkUpdateOptions*

OleIsDcMeta 3.1

Syntax
```
#include <ole.h>
BOOL OleIsDcMeta(hdc)

HDC hdc;            /* Handle des Gerätekontexts      */
```

Beschreibung Diese Funktion ermittelt, ob der angegebene Gerätekontext ein Metafile-Gerätekontext ist.

Parameter *hdc:* Bezeichnet den abzufragenden Gerätekontext.

Rückgabewert Der Rückgabewert ist positiv, wenn der Gerätekontext ein Metafile-Gerätekontext ist. Andernfalls ist er *NULL*.

OleLoadFromStream 3.1

Syntax
```
#include <ole.h>
OLESTATUS OleLoadFromStream(lpStream, lpszProtocol,
  lpClient, lhClientDoc, lpszObjname, lplpObject)

LPOLESTREAM lpStream;        /* Adresse des Stream
                                für das Objekt       */
LPCSTR lpszProtocol;         /* Adresse des String für
                                den Protokollnamen   */
LPOLECLIENT lpClient;        /* Adresse der Client-
                                Struktur             */
LHCLIENTDOC lhClientDoc;     /* long-Handle des
                                Client-Dokuments     */
LPCSTR lpszObjname;          /* Adresse des String
                                für den Objektnamen  */
LPOLEOBJECT FAR* lplpObject; /* Adresse des Zeigers
                                auf das Objekt       */
```

Beschreibung Diese Funktion lädt ein Objekt aus dem Dokument, das das Objekt enthält.

Parameter *lpStream:* Zeigt auf eine *OLESTREAM*-Struktur, deren Speicherplatz von der Client-Anwendung reserviert und initialisiert worden ist. Die Bibliothek ruft die Funktion *Get* der *OLESTREAMVTBL*-Struktur auf, um die Daten für das Objekt zu erhalten.

lpszProtocol: Zeigt auf einen nullterminierten String, der den Namen des benötigten Protokolls bestimmt. Zur Zeit kann dieser Wert StdFileEditing (der Name des OLE-Protokolls) oder Static für nicht bearbeitbare Bilder sein.

lpClient: Zeigt auf eine *OLECLIENT*-Struktur, deren Speicherplatz von der Client-Anwendung reserviert und initialisiert wurde. Dieser Zeiger wird verwendet, um die richtige Callback-Funktion aufzufinden. Er wird in Callback-Benachrichtigungen übergeben.

lhClientDoc: Bezeichnet das Client-Dokument, in dem das Objekt erzeugt wird.

lpszObjname: Zeigt auf einen nullterminierten String, der den Namen des Client für das Objekt bestimmt.

lplpObject: Zeigt auf eine Variable, in der die Bibliothek einen Zeiger auf das geladene Objekt speichert.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_HANDLE
OLE_ERROR_NAME
OLE_ERROR_PROTOCOL
OLE_ERROR_STREAM
OLE_WAIT_FOR_RELEASE

Hinweis Um ein Objekt zu laden, benötigt die Client-Anwendung nur die Position des Objekts in einer Datei. Normalerweise lädt ein Client ein Objekt nur, wenn das Objekt benötigt wird, weil es zum Beispiel angezeigt werden muß. Wenn ein Objekt nicht geladen werden kann, während der Parameter *lpszProtocol* StdFileEditing enthält, kann eine Anwendung die Funktion *OleLoadFromStream* erneut aufrufen und dabei Static angeben.

Ist ein Objekt verknüpft und der Server wie auch das Dokument geöffnet, verknüpft die Bibliothek Client und Server beim Aufruf der Funktion *OleLoadFromStream* automatisch.

Siehe auch *OleQuerySize*,*OleSaveToStream*

OleLockServer 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleLockServer(lpObject, lphServer)

LPOLEOBJECT lpObject;    /* Adresse des Objekts       */
LHSERVER FAR* lphServer; /* Adresse des Server-Handle */
```

Beschreibung Diese Funktion wird von Client-Anwendungen aufgerufen, um eine geöffnete Server-Anwendung im Speicher zu halten.

Dies erlaubt es der Client-Bibliothek, die Server-Anwendung zum schnellen Öffnen von Objekten zu benutzen.

Parameter

lpObject: Zeigt auf ein Objekt, mit dem die Client-Bibliothek die geöffnete Server-Anwendung bezeichnet, die im Speicher gehalten werden soll. Wenn der Server gesperrt wurde, kann das Objekt gelöscht werden.

lphServer: Zeigt auf ein Handle der Server-Anwendung, wenn die Funktion zurückkehrt.

Rückgabewert

Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_COMM
OLE_ERROR_LAUNCH
OLE_ERROR_OBJECT

Hinweis

Ein Client ruft die Funktion *OleLockServer* auf, um das Öffnen von Objekten zu beschleunigen, wenn derselbe Server für eine Reihe verschiedener Objekte benutzt wird. Bevor der Client beendet wird, muß er die Funktion *OleUnlockServer* aufrufen, um den Server aus dem Speicher freizugeben.

Wird die Funktion *OleLockServer* mehrmals für einen bestimmten Server aufgerufen – sei es auch durch verschiedene Client-Anwendungen – erhöht sich der Sperrenzähler des Server. Jeder Aufruf von *OleUnlockServer* verringert ihn wieder. Der Server bleibt gesperrt, bis der Zähler den Wert Null hat. Wird das vom Parameter *lpObject* bezeichnete Objekt gelöscht, bevor die Funktion *OleUnlockServer* aufgerufen wird, muß die Funktion *OleUnlockServer* dennoch aufgerufen werden, um den Sperrenzähler zu verringern.

Falls notwendig, kann ein Server trotz eines Client-Aufrufs von *OleLockServer* beendet werden.

Siehe auch

OleUnlockServer

OleObjectConvert 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleObjectConvert(lpObject, lpszProtocol,
  lpClient, lhClientDoc, lpszObjname, lplpObject)

LPOLEOBJECT lpObject;        /* Adresse des zu konver-
                                tierenden Objekts     */
LPCSTR lpszProtocol;         /* Adresse des String für
                                den Protokollnamen    */
LPOLECLIENT lpClient;        /* Adresse des Client
                                für das neue Objekt   */
```

```
LHCLIENTDOC lhClientDoc;     /* long-Handle des
                                Client-Dokuments     */
LPCSTR lpszObjname;          /* Adresse des String
                                für den Objektnamen  */
LPOLEOBJECT FAR* lplpObject; /* Adresse des Zeigers
                                auf das neue Objekt  */
```

Beschreibung Diese Funktion erzeugt ein neues Objekt, das ein bestimmtes Protokoll unterstützt, indem es ein bestehendes Objekt konvertiert. Diese Funktion löscht weder die Orginaldatei noch ersetzt sie diese.

Parameter *lpObject:* Zeigt auf das zu konvertierende Objekt.

lpszProtocol: Zeigt auf einen nullterminierten String, der den Namen des benötigten Protokolls bestimmt. Zur Zeit kann dieser Wert Static für nicht bearbeitbare Bilder sein.

lpClient: Zeigt auf eine *OLECLIENT*-Struktur für das neue Objekt.

lhClientDoc: Bezeichnet das Client-Dokument, in dem das Objekt erzeugt wird.

lpszObjname: Zeigt auf einen nullterminierten String, der den Namen des Client für das Objekt bestimmt. Dieser Name muß hinsichtlich der Namen aller anderen Objekte im Dokument eindeutig sein und darf keinen Schrägstrich (/) enthalten.

lplpObjname: Zeigt auf eine Variable, in der die Bibliothek einen Zeiger auf das neue Objekt speichert.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_BUSY
OLE_ERROR_HANDLE
OLE_ERROR_NAME
OLE_ERROR_OBJECT
OLE_ERROR_STATIC

Hinweis Die einzige Konvertierung, die gegenwärtig unterstützt wird, ist das Verändern eines verknüpften oder eingebetteten Objekts in ein statisches Objekt.

Siehe auch *OleClone*

OleQueryBounds 3.1

Syntax
```
#include <ole.h>
OLESTATUS OleQueryBounds(lpObject, lpBounds)
```

```
LPOLEOBJECT lpObject;  /* Adresse des abzu-
                          fragenden Objekts          */
RECT FAR* lpBounds;    /* Adresse der Struktur für
                          das umgebende Rechteck     */
```

Beschreibung Diese Funktion ruft die Abmessungen des umgebenden Rechtecks auf dem Zielgerät für das angebende Objekt ab. Die Koordinaten sind in *MM_HIMETRIC*-Einheiten angegeben.

Parameter *lpObject:* Zeigt auf das abzufragende Objekt.

lpBounds: Zeigt auf eine *RECT*-Struktur für die Abmessungen des umgebenden Rechtecks. Die Elemente dieser Struktur haben folgende Bedeutung:

Element	Bedeutung
rect.left	0
rect.top	0
rect.right	Abmessung in x-Richtung
rect.bottom	Abmessung in y-Richtung

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_BLANK
OLE_ERROR_MEMORY
OLE_ERROR_OBJECT

Siehe auch *OleSetBounds*, *SetMapMode*

OleQueryClientVersion 3.1

Syntax
```
#include <ole.h>
DWORD OleQueryClientVersion(void)
```

Beschreibung Diese Funktion ruft die Versionsnummer der Client-Bibliothek ab.

Parameter Diese Funktion besitzt keine Parameter.

Rückgabewert Der Rückgabewert ist ein *DWORD*-Wert. Die größere Hauptversionsnummer befindet sich im niederwertigen Byte des niederwertigen *WORD*, die Nebenversionsnummer im höherwertigen Byte des niederwertigen *WORD*. Das höherwertige *WORD* ist reserviert.

Siehe auch *OleQueryServerVersion*

OleQueryCreateFromClip 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleQueryCreateFromClip(lpszProtocol,
   renderopt, cfFormat)

LPCSTR lpszProtocol;      /* Adresse des String
                             für den Protokollnamen   */
OLEOPT_RENDER renderopt;  /* übergabeoptionen         */
OLECLIPFORMAT cfFormat;   /* Format der Zwischen-
                             ablagedaten              */
```

Beschreibung

Diese Funktion überprüft, ob das Objekt in der Zwischenablage das angegebene Protokoll und die angegebenen Übergabeoptionen unterstützt.

Parameter

lpszProtocol: Zeigt auf einen nullterminierten String, der den Namen des Protokolls enthält, das der Client benötigt. Zur Zeit kann dieser Wert StdFileEditing (der Name des OLE-Protokolls) oder Static für nicht bearbeitbare Bilder sein.

renderopt: Bestimmt die bevorzugten Präsentationsdaten des Client. Dieser Parameter kann folgende Werte annehmen:

Wert	Bedeutung
olerender_draw	Der Client ruft die Funktion *OleDraw* auf. Die Bibliothek erhält und verwaltet die Präsentationsdaten.
olerender_format	Die Bibliothek erhält und verwaltet die Daten in dem Format, das im Parameter *cfFormat* angegeben wurde.
olerender_none	Die Client-Bibliothek erhält keine Präsentationsdaten und zeichnet das Objekt nicht.

cfFormat: Bestimmt das Zwischenablageformat. Dieser Parameter wird nur verwendet, wenn der Parameter *renderopt* den Wert *olerender_format* hat. Ist das Zwischenablageformat *CF_METAFILEPICT*, *CF_DIB* oder *CF_BITMAP*, verwaltet die Bibliothek die Daten und zeichnet das Objekt. Die Bibliothek unterstützt das Zeichnen in keinem anderen Format.

Rückgabewert

Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_FORMAT
OLE_ERROR_PROTOCOL

Hinweis

Die Funktion *OleQueryCreateFromClip* wird normalerweise verwendet, um zu prüfen, ob der Befehl *Einfügen* aktiviert werden

soll. Die Übergabeoption *olerender_none* wird normalerweise verwendet, um hypertextartige Verbindungen zu unterstützen. Mit dieser Option ruft der Client nicht die Funktion *OleDraw* und die Funktion *OleGetData* nur für ObjectLink-, OwnerLink- und Native-Formate auf.

Die Übergabeoption *olerender_format* erlaubt einem Client, Daten zu berechnen anstatt sie zu zeichnen. Außerdem kann er ungewöhnliche Datenformate benutzen oder ein Standarddatenformat verändern. Mit dieser Option ruft der Client nicht die Funktion *OleDraw* auf. Sein Aufruf von *OleGetData* liefert die Daten im angegebenen Format zurück.

Die gebräuchlichste Übergabeoption ist *olerender_draw*. Sie ist die für den Client am leichtesten zu implementierende Übergabeoption (der Client muß nur *OleDraw* aufrufen), und sie ist sehr flexibel in der Verwendung. Ein Objekt-Handler kann diese Flexibilität ausnutzen, um entweder überhaupt keine Präsentationsdaten, ein privates Datenformat oder mehrere verschiedene Formate, zwischen denen frei gewählt werden kann, zu speichern. Zukünftige Implementierungen von OLE werden die Flexibilität dieser Option weitergehend ausnutzen.

Siehe auch *OleCreateFromClip*, *OleDraw*, *OleGetData*

OleQueryLinkFromClip 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleQueryLinkFromClip(lpszProtocol,
   renderopt, cfFormat)

LPCSTR lpszProtocol;      /* Adresse des String für
                             den Protokollnamen       */
OLEOPT_RENDER renderopt;  /* übergabeoptionen         */
OLECLIPFORMAT cfFormat;   /* Format der Zwischen-
                             ablagedaten              */
```

Beschreibung Diese Funktion überprüft, ob eine Client-Anwendung die Daten in der Zwischenablage verwenden kann, um ein verknüpftes Objekt zu erzeugen, das das angegebene Protokoll und die Übergabeoptionen unterstützt.

Parameter *lpszProtocol:* Zeigt auf einen nullterminierten String, der den Namen des vom Client benötigten Protokolls bestimmt. Dieser Wert kann StdFileEditing sein, der Name des OLE-Protokolls.

renderopt: Bestimmt die bevorzugten Präsentationsdaten des Client. Dieser Parameter kann einen der in der folgenden Tabelle beschriebenen Werte annehmen:

Wert	Bedeutung
olerender_draw	Der Client ruft *OleDraw* auf. Die Bibliothek erhält und verwaltet die Präsentationsdaten.
olerender_format	Die Bibliothek erhält und verwaltet die Daten im angeforderten Format, das im Parameter *cfFormat* bestimmt wird.
olerender_none	Die Client-Bibliothek erhält keine Präsentationsdaten und zeichnet das Objekt nicht.

cfFormat: Bestimmt das Zwischenablageformat. Dieser Parameter wird nur verwendet, wenn der Parameter *renderopt* den Wert *olerender_format* enthält. Ist das Zwischenablageformat *CF_METAFILEPICT*, *CF_DIB* oder *CF_BITMAP*, verwaltet die Bibliothek die Daten und zeichnet das Objekt. Die Bibliothek unterstützt das Zeichnen in keinem anderen Format.

Rückgabewert

Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_FORMAT
OLE_ERROR_PROTOCOL

Hinweis

Die Funktion *OleQueryLinkFromClip* wird normalerweise verwendet, um zu prüfen, ob der Befehl *Verknüpfung einfügen* aktiviert werden soll.

Die Übergabeoption *olerender_none* wird normalerweise verwendet, um hypertextartige Verbindungen zu unterstützen. Mit dieser Option ruft der Client nicht die Funktion *OleDraw* und die Funktion *OleGetData* nur für ObjectLink-, OwnerLink- und Native-Formate auf.

Die Übergabeoption *olerender_format* erlaubt einem Client, Daten zu berechnen anstatt sie zu zeichnen. Außerdem kann er ungewöhnliche Datenformate benutzen oder ein Standarddatenformat verändern. Mit dieser Option ruft der Client nicht die Funktion *OleDraw* auf. Sein Aufruf von *OleGetData* liefert die Daten im angegebenen Format zurück.

Die gebräuchlichste Übergabeoption ist *olerender_draw*. Sie ist die für den Client am leichtesten zu implementierende Übergabeoption (der Client muß nur *OleDraw* aufrufen), und sie ist sehr flexibel in der Verwendung. Ein Objekt-Handler kann diese Flexibilität ausnutzen, um entweder überhaupt keine Präsentationsdaten, ein privates Datenformat oder mehrere verschiedene Formate, zwischen denen frei gewählt werden kann, zu speichern. Zukünftige Implementierungen von OLE werden die Flexibilität dieser Option ebenfalls ausnutzen.

Siehe auch

OleCreateFromClip, *OleDraw*, *OleGetData*

OleQueryName 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleQueryName(lpObject, lpszObject,
   lpwBuffSize)

LPOLEOBJECT lpObject;   /* Adresse des Objekts       */
LPCSTR lpszObject;      /* Adresse des String für
                           den Objektnamen           */
UINT FAR* lpwBuffSize;  /* Adresse des WORD für
                           die Puffergröße           */
```

Beschreibung Diese Funktion ruft den Namen des angegebenen Objekts ab.

Parameter *lpObject:* Zeigt auf das Objekt, dessen Name abgefragt wird.

lpszObject: Zeigt auf ein Zeichen-Array, das einen nullterminierten String enthält. Wenn die Funktion zurückkehrt, enthält der String den Namen des Objekts.

lpwBuffSize: Zeigt auf eine Variable, die die Größe des Puffers in Byte enthält, auf den der Parameter *lpszObject* zeigt.Wenn die Funktion zurückkehrt, ist der Wert die Anzahl Byte, die in den Puffer kopiert worden sind.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls liefert die Funktion einen Fehlerwert, der *OLE_ERROR_OBJECT* sein kann.

Siehe auch *OleRename*

OleQueryOpen 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleQueryOpen(lpObject)

LPOLEOBJECT lpObject;      /* Adresse des abzu-
                              fragenden Objekts      */
```

Beschreibung Diese Funktion überprüft, ob das angegebene Objekt geöffnet ist.

Parameter *lpObject:* Zeigt auf das abzufragende Objekt.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn das Objekt geöffnet ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_COMM
OLE_ERROR_OBJECT
OLE_ERROR_STATIC

Siehe auch *OleActivate*

OleQueryOutOfDate 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleQueryOutOfDate(lpObject)

LPOLEOBJECT lpObject;        /* Adresse des abzu-
                                fragenden Objekts     */
```

Beschreibung Diese Funktion überprüft, ob ein Objekt veraltet ist.

Parameter *lpObject:* Zeigt auf das abzufragende Objekt.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn das Objekt aktuell ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_OBJECT
OLE_ERROR_OUTOFDATE

Hinweis Die Funktion *OleQueryOutOfDate* ist nicht für die aktuelle OLE-Version implementiert worden. Für verknüpfte Objekte liefert die Funktion immer den Wert *OLE_OK*.

Ein verknüpftes Objekt könnte veraltet sein, wenn das Dokument, das die Quelle der Verknüpfung ist, aktualisiert worden ist. Ein eingebettetes Objekt, das Verknüpfungen zu anderen Objekten besitzt, könnte ebenfalls veraltet sein.

Siehe auch *OleEqual*, *OleUpdate*

OleQueryProtocol 3.1

Syntax

```
#include <ole.h>
void FAR* OleQueryProtocol(lpobj, lpszProtocol)

LPOLEOBJECT lpobj;     /* Adresse des abzufragenden
                          Objekts                     */
LPCSTR lpszProtocol;   /* Adresse des String für
                          das abzufragende Protokoll  */
```

Beschreibung Diese Funktion überprüft, ob ein Objekt ein bestimmtes Protokoll unterstützt.

Parameter *lpobj:* Zeigt auf das abzufragende Objekt.

lpszProtocol: Zeigt auf einen nullterminierten String, der den Namen des verlangten Protokolls bestimmt. Dieser Wert kann StdFileEditing oder StdExecute sein.

Rückgabewert Der Rückgabewert ist ein **void**-Zeiger auf eine *OLEOBJECT*-Struktur, wenn die Funktion erfolgreich ist. Er ist *NULL*, wenn das Objekt das angeforderte Protokoll nicht unterstützt. Die Bibliothek kann *OLE_WAIT_FOR_RELEASE* zurückliefern, wenn eine Anwendung die Funktion aufruft.

Hinweis Die Funktion *OleQueryProtocol* überprüft, ob das angegebene Protokoll unterstützt wird und gibt einen veränderten Objektzeiger zurück, der den Zugriff auf die Funktionstabelle für das Protokoll ermöglicht. Dieser veränderte Objektzeiger zeigt auf eine Struktur, die dieselbe Form wie die *OLEOBJECT*-Struktur besitzt. Die neue Struktur zeigt auf eine Funktionstabelle und kann zusätzliche Statusinformationen enthalten. Der neue Zeiger zeigt nicht auf ein anderes Objekt – falls das Objekt gelöscht wird, werden sekundäre Zeiger ungültig. Enthält ein Protokoll Löschfunktionen, macht der Aufruf einer Löschfunktion alle Zeiger auf das Objekt ungültig.

Eine Client-Anwendung ruft normalerweise *OleQueryProtocol* auf und gibt StdExecute im Parameter *lpszProtocol* an, bevor die Funktion *OleExecute* aufgerufen wird. Dies erlaubt der Client-Anwendung zu prüfen, ob der Server für ein Objekt DDE-Execute-Befehle unterstützt.

Siehe auch *OleExecute*

OleQueryReleaseError 3.1

Syntax
```
#include <ole.h>
OLESTATUS OleQueryReleaseError(lpobj)

LPOLEOBJECT lpobj;       /* Adresse des abzu-
                            fragenden Objekts        */
```

Beschreibung Diese Funktion überprüft den Fehlerwert für eine asynchrone Operation an einem Objekt.

Parameter *lpobj:* Zeigt auf das Objekt, für das der Fehlerwert abgefragt wird.

Rückgabewert Ist die Funktion erfolgreich, so ist der Rückgabewert entweder *OLE_OK*, wenn die asynchrone Operation erfolgreich durchgeführt wurde, oder der Fehlerwert dieser Operation. Falls der Zeiger ungültig ist, der im Parameter *lpobj* übergeben wird, liefert die Funktion *OLE_ERROR_OBJECT* zurück.

Hinweis Eine Client-Anwendung erhält die Benachrichtigung *OLE_RELEASE*, wenn eine asynchrone Operation abgeschlossen worden ist. Der Client sollte dann *OleQueryReleaseError* aufrufen, um zu prüfen, ob die Operation erfolgreich oder mit einem Fehlerwert abgeschlossen worden ist.

Siehe auch *OleQueryReleaseMethod*, *OleQueryReleaseStatus*

OleQueryReleaseMethod 3.1

Syntax
```
#include <ole.h>
OLE_RELEASE_METHOD OleQueryReleaseMethod(lpobj)
```

```
LPOLEOBJECT lpobj;    /* Adresse des abzufragenden
                         Objekts                    */
```

Beschreibung Diese Funktion ermittelt die Operation, die am angegebenen Objekt ausgeführt worden ist.

Parameter *lpobj:* Zeigt auf das Objekt, für das die Operation abgefragt werden soll.

Rückgabewert Der Rückgabewert zeigt die Server-Operation (Methode) an, die ausgeführt worden ist. Er kann einen der folgenden Werte annehmen:

Wert	Server-Operation
OLE_ACTIVE	Aktivieren
OLE_CLOSE	Schließen
OLE_COPYFROMLINK	Aus Verknüpfung kopieren und Verbindung wiederaufnehmen
OLE_CREATE	Erzeugen
OLE_CREATEFROMFILE	Erzeugen aus Datei
OLE_CREATEFROMTEMPLATE	Erzeugen aus Schablone
OLE_CREATEINVISIBLE	Verborgen erzeugen
OLE_CREATELINKFROMFILE	Verknüpfung aus Datei erzeugen
OLE_DELETE	Objekt löschen
OLE_EMBPASTE	Einfügen und aktualisieren
OLE_LINKPASTE	Verknüpfung einfügen und Verbindung wiederaufnehmen
OLE_LOADFROMSTREAM	Aus Stream laden und Verbindung wiederaufnehmen
OLE_NONE	Keine Operation aktiv
OLE_OTHER	Verschiedene andere asynchrone Operationen
OLE_RECONNECT	Wiederaufnahme der Verbindung
OLE_REQUESTDATA	*OleRequestData*
OLE_RUN	Starten
OLE_SERVERUNLAUNCH	Der Server stoppt
OLE_SETDATA	*OleSetData*
OLE_SETUPDATEOPTIONS	Setzten der Aktualisierungsoptionen

Wert	Server-Operation
OLE_SHOW	Anzeigen
OLE_UPDATE	Aktualisieren

Wenn der im Parameter *lpobj* übergebene Zeiger ungültig ist, liefert die Funktion den Wert *OLE_ERROR_OBJECT*.

Hinweis — Eine Client-Anwendung erhält die Benachrichtigung *OLE_RELEASE*, wenn eine asynchrone Operation abgeschlossen worden ist. Der Client ruft anschließend *OleQueryReleaseMethod* auf, um zu prüfen, ob die Operation die Bibliothek veranlaßt hat, die Benachrichtigung *OLE_RELEASE* zu senden. Der Client ruft *OleQueryReleaseError* auf, um zu bestimmen, ob die Operation erfolgreich oder mit einem Fehlerwert abgeschlossen worden ist.

Siehe auch — *OleQueryReleaseError*, *OleQueryReleaseStatus*

OleQueryReleaseStatus 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleQueryReleaseStatus(lpobj)

LPOLEOBJECT lpobj;          /* Adresse des abzu-
                               fragenden Objekts     */
```

Beschreibung — Diese Funktion ermittelt, ob eine Operation für das angegebene Objekt ausgeführt worden ist.

Parameter — *lpobj:* Zeigt auf das Objekt, für das die Operation abgefragt wird.

Rückgabewert — Wenn die Funktion erfolgreich ist, wird *OLE_OK* oder *OLE_BUSY* – wenn die Operation noch bearbeitet wird – zurückgegeben. Ist der im Parameter *lpObj* übergebene Zeiger ungültig, gibt die Funktion *OLE_ERROR_OBJECT* zurück.

Siehe auch — *OleQueryReleaseError*, *OleQueryReleaseMethod*

OleQueryServerVersion 3.1

Syntax

```
#include <ole.h>
DWORD OleQueryServerVersion(void)
```

Beschreibung — Diese Funktion ruft die Versionsnummer der Server-Bibliothek ab.

Parameter — Diese Funktion hat keine Parameter.

Rückgabewert — Der Rückgabewert ist ein *DWORD*-Wert. Die Hauptversionsnummer befindet sich im niederwertigen Byte des niederwer-

tigen *WORD*, die Unterversionsnummer im höherwertigen Byte des niederwertigen *WORD*. Das höherwertige *WORD* ist reserviert.

Siehe auch *OleQueryClientVersion*

OleQuerySize 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleQuerySize(lpObject, pdwSize)

LPOLEOBJECT lpObject;  /* Adresse des abzufragenden
                          Objekts                    */
DWORD FAR* pdwSize;    /* Adresse der Objektgröße    */
```

Beschreibung Diese Funktion liefert die Größe des angegebenen Objekts zurück.

Parameter *lpObject:* Zeigt auf das abzufragende Objekt.

pdwSize: Zeigt auf eine Variable für die Größe des Objekts. Diese Variable enthält die Größe des Objekts, wenn die Funktion zurückkehrt.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_BLANK
OLE_ERROR_MEMORY
OLE_ERROR_OBJECT

Siehe auch *OleLoadFromStream*

OleQueryType 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleQueryType(lpObject, lpType)

LPOLEOBJECT lpObject;   /* Adresse des abzufragenden
                           Objekts                   */
LONG FAR* lpType;       /* Adresse des Objekttypen   */
```

Beschreibung Diese Funktion überprüft, ob ein angegebenes Objekt eingebettet, verknüpft oder statisch ist.

Parameter *lpObject:* Zeigt auf das Objekt, dessen Typ abgefragt werden soll.

lpType: Zeigt auf eine **long**-Variable, die den Typ des Objekts enthält, wenn die Funktion zurückkehrt. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
OT_EMBEDDED	Das Objekt ist eingebettet.
OT_LINK	Das Objekt ist verknüpft.
OT_STATIC	Das Objekt ist ein statisches Bild.

Rückgabewert Der Rückgabewert ist *OLE_OK,* wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_GENERIC
OLE_ERROR_OBJECT

Siehe auch *OleEnumFormats*

OleReconnect 3.1

Syntax
```
#include <ole.h>
OLESTATUS OleReconnect(lpObject)

LPOLEOBJECT lpObject;  /* Adresse des Objekts, das
                          wiederverbunden werden soll */
```

Beschreibung Diese Funktion stellt eine Verknüpfung für ein geöffnetes verknüpftes Objekt wieder her. Wenn das angegebene Objekt noch nicht geöffnet ist, öffnet diese Funktion es nicht.

Parameter *lpObject:* Zeigt auf das Objekt, mit dem die Verbindung wiederhergestellt werden soll.

Rückgabewert Der Rückgabewert ist *OLE_OK,* wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_BUSY
OLE_ERROR_NOT_LINK
OLE_ERROR_OBJECT
OLE_ERROR_STATIC
OLE_WAIT_FOR_RELEASE

Hinweis Eine Client-Anwendung kann *OleReconnect* verwenden, um die Präsentation für ein verknüpftes Ojekt aktuell zu halten.

Siehe auch *OleActivate, OleClose, OleUpdate*

OleRegisterClientDoc 3.1

Syntax
```
#include <ole.h>
OLESTATUS OleRegisterClientDoc(lpszClass, lpszDoc,
   reserved, lplhDoc)
```

```
LPCSTR lpszClass;          /* Adresse des String
                              für den Klassennamen    */
LPCSTR lpszDoc;            /* Adresse des String für
                              den Namen des Dokuments */
LONG reserved;             /* reserviert              */
LHCLIENTDOC FAR* lplhDoc;  /* Adresse des Dokumenten-
                              Handle                  */
```

Beschreibung Diese Funktion registriert ein geöffnetes Client-Dokument für die Bibliothek und liefert das Handle des Dokuments zurück.

Parameter *lpszClass:* Zeigt auf einen nullterminierten String, der die Klasse des Client-Dokuments bestimmt.

lpszDoc: Zeigt auf einen nullterminierten String, der die Position des Client-Dokuments bestimmt. Dieser Wert sollte ein vollständiger Pfadname sein.

reserved: Ist reserviert und muß Null sein.

lplhDoc: Zeigt auf das Handle des Client-Dokuments, wenn die Funktion zurückkehrt. Dieses Handle wird benutzt, um das Dokument in anderen Funktionen der Dokumentenverwaltung zu bezeichnen.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich war. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_ALREADY_REGISTERED
OLE_ERROR_MEMORY
OLE_ERROR_NAME

Hinweis Wenn ein in die Zwischenablage kopiertes Objekt nur existiert, weil die Client-Anwendung Native-Daten kopiert, die Objekte enthalten, muß der Parameter *lpszDoc* den Namen Clipboard enthalten.

Client-Anwendungen sollten geöffnete Dokumente für die Bibliothek registrieren und die Bibliothek benachrichtigen, wenn ein Dokument umbenannt, gesichert oder in einem veränderten Zustand wiederhergestellt wird.

Siehe auch *OleRenameClientDoc, OleRevertClientDoc, OleRevokeClientDoc, OleSavedClientDoc*

OleRegisterServer 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleRegisterServer(lpszClass, lpsrvr,
    lplhserver, hinst, srvuse)

LPCSTR lpszClass;          /* Adresse des Klassen-
                              namen-String            */
```

```
LPOLESERVER lpsrvr;        /* Adresse der OLESERVER-
                              Struktur                */
LHSERVER FAR* lplhserver;  /* Adresse des Server-
                              Handle                  */
HINSTANCE hinst;           /* Handle der Instanz      */
OLE_SERVER_USE srvuse;     /* eine oder mehrere
                              Instanzen               */
```

Beschreibung *OleRegisterServer* registriert den angegebenen Server, Klassennamen und die Instanz für die Server-Bibliothek.

Parameter *lpszClass:* Zeigt auf einen nullterminierten String mit dem Klassennamen, der registriert werden soll.

lpsrvr: Zeigt auf eine von der Server-Anwendung reservierte und initialisierte *OLESERVER*-Struktur.

lplhserver: Zeigt auf eine Variable vom Typ *LHSERVER*, in der die Bibliothek das Handle des Server speichert. Dieses Handle wird zum Beispiel in den Funktionen *OleRegisterServerDoc* und *OleRevokeServer* verwendet.

hinst: Bezeichnet die Instanz der Server-Anwendung. Dieses Handle stellt sicher, daß Client-Anwendungen mit der richtigen Instanz einer Server-Anwendung eine Verbindung einrichten.

srvuse: Bestimmt, ob der Server für die Bearbeitung mehrerer Objekte eine einzelne oder mehrere Instanzen verwendet. Dieser Wert muß entweder *OLE_SERVER_SINGLE* oder *OLE_SERVER_MULTI* sein.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_CLASS
OLE_ERROR_MEMORY
OLE_ERROR_PROTECT_ONLY

Hinweis Wenn die Server-Anwendung startet, erzeugt sie eine *OLESERVER*-Struktur und ruft die Funktion *OleRegisterServer* auf. Unterstützt ein Server mehrere Klassennamen, kann er für jeden Namen eine Struktur reservieren oder die gleiche Struktur mehrmals benutzen. Der Klassenname wird an Server-Funktionen übergeben, die von der Bibliothek aufgerufen werden. Dadurch können Server die angeforderte Klasse ermitteln, wenn sie mehrere Klassen unterstützen.

Der Parameter *srvuse* wird von Bibliotheken beim Öffnen eines Objekts verwendet. Wenn er den Wert *OLE_SERVER_MULTI* hat und alle aktuellen Instanzen bereits ein Objekt bearbeiten, wird eine neue Instanz der Server-Anwendung gestartet. Server, die die Multidokumentschnittstelle (MDI)

unterstützen, geben normalerweise den Wert *OLE_SERVER_-SINGLE* an.

Siehe auch *OleRegisterServerDoc, OleRevokeServer*

OleRegisterServerDoc 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleRegisterServerDoc(lhsrvr, lpszDocName,
    lpdoc, lplhdoc)

LHSERVER lhsrvr;          /* Handle der Server-
                             Anwendung            */
LPCSTR lpszDocName;       /* Adresse des Dokument-
                             namen-String         */
LPOLESERVERDOC lpdoc;     /* Adresse der OLESERVERDOC-
                             Struktur             */
LHSERVERDOC FAR* lplhdoc; /* Handle des registrierten
                             Dokuments            */
```

Beschreibung Die Funktion *OleRegisterServerDoc* registriert ein Dokument für die Server-Bibliothek, wenn andere Client-Anwendungen mit ihm verknüpft sind. Ein Server ruft diese Funktion auf, wenn er mit der Option */Embedding Dateiname* gestartet wird oder wenn er ein Dokument erzeugt oder öffnet, das nicht von der Bibliothek angefordert wurde.

Parameter *lhsrvr:* Bezeichnet den Server. Server-Anwendungen können dieses Handle durch die Funktion *OleRegisterServer* ermitteln.

lpszDocName: Zeigt auf einen nullterminierten String, der den Namen des Dokuments enthält. Dieser Parameter sollte ein vollständiger Pfadname sein.

lpdoc: Zeigt auf eine von der Server-Anwendung reservierte und initialisierte *OLESERVERDOC*-Struktur.

lplhdoc: Zeigt nach Beendigung der Funktion auf das Handle des Dokuments.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_ADDRESS
OLE_ERROR_HANDLE
OLE_ERROR_MEMORY

Hinweis Wenn das Dokument auf Anforderung der Server-Bibliothek erzeugt oder geöffnet worden ist, sollte der Server das Dokument nicht mit der Funktion *OleRegisterServerDoc* registrieren, sondern durch den Parameter einen Zeiger auf die *OLESERVER-DOC*-Struktur an die entsprechende Funktion zurückgeben.

Siehe auch *OleRegisterServer, OleRevokeServerDoc*

OleRelease 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleRelease(lpObject)

LPOLEOBJECT lpObject;  /* Adresse des freizugebenden
                          Objekts                    */
```

Beschreibung Die Funktion *OleRelease* gibt den Speicher eines Objekts frei und schließt es, wenn es geöffnet war. Sie zeigt nicht an, daß das Objekt aus dem Client-Dokument gelöscht worden ist.

Parameter *lpObject:* Zeigt auf das Objekt, das freigegeben werden soll.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_BUSY
OLE_ERROR_OBJECT
OLE_WAIT_FOR_RELEASE

Hinweis Die Funktion *OleRelease* sollte für alle Objekte aufgerufen werden, wenn das Client-Dokument geschlossen wird.

Siehe auch *OleDelete*

OleRename 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleRename(lpObject, lpszNewname)

LPOLEOBJECT lpObject;  /* Adresse des Objekts        */
LPCSTR lpszNewname;    /* Adresse des String mit
                          dem neuen Objektnamen      */
```

Beschreibung Die Funktion *OleRename* benennt ein Objekt um.

Parameter *lpObject:* Zeigt auf das Objekt, das umbenannt werden soll.

lpszNewname: Zeigt auf einen nullterminierten String, der den neuen Namen des Objekts enthält.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann der Fehlerwert *OLE_ERROR_OBJECT* zurückgegeben werden.

Hinweis Der Benutzer benötigt die Objektnamen nicht. Sie müssen innerhalb des Dokuments eindeutig sein und mit ihm gespeichert werden.

Siehe auch *OleQueryName*

OleRenameClientDoc 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleRenameClientDoc(lhClientDoc,
   lpszNewDocname)

LHCLIENTDOC lhClientDoc;  /* Handle des Client-
                             Dokuments              */
LPCSTR lpszNewDocname;    /* Adresse des String mit
                             dem neuen Objektnamen  */
```

Beschreibung Die Funktion *OleRenameClientDoc* informiert die Client-Bibliothek, daß ein Dokument umbenannt worden ist. Eine Client-Anwendung ruft diese Funktion auf, wenn ein Dokumentname geändert wird – z.B. wenn der Benutzer den Befehl *Speichern* oder *Speichern unter* aus dem Menü *Datei* wählt.

Parameter *lhClientDoc:* Bezeichnet das umbenannte Dokument.

lpszNewDocname: Zeigt auf einen nullterminierten String, der den neuen Namen des Dokuments angibt.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann der Fehlerwert *OLE_ERROR_HANDLE* zurückgegeben werden.

Hinweis Client-Anwendungen sollten geöffnete Dokumente für die Bibliothek registrieren und sie benachrichtigen, wenn ein Dokument umbenannt, geschlossen, gespeichert oder in einem geänderten Zustand wiederhergestellt wird.

Siehe auch *OleRegisterClientDoc*, *OleRevertClientDoc*, *OleRevokeClientDoc*, *OleSavedClientDoc*

OleRenameServerDoc 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleRenameServerDoc(lhDoc, lpszDocName)

LHSERVERDOC lhDoc;      /* Handle des Dokuments      */
LPCSTR lpszDocName;     /* Adresse des Pfad- und
                           Dateinamen-String         */
```

Beschreibung Die Funktion *OleRenameServerDoc* informiert die Server-Bibliothek, daß ein Dokument umbenannt worden ist.

Parameter *lhDoc:* Bezeichnet das umbenannte Dokument.

lpszDocName: Zeigt auf einen nullterminierten String, der den neuen Namen des Dokuments angibt. Dieser Parameter ist normalerweise ein vollständiger Pfadname.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist.

Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_HANDLE
OLE_ERROR_MEMORY

Hinweis

Ein Aufruf der Funktion *OleRenameServerDoc* entspricht dem Senden der Benachrichtigung *OLE_RENAMED* an die Callback-Funktion der Client-Anwendung. Die Server-Anwendung ruft diese Funktion auf, wenn sie ein Dokument umbenennt, mit dem die aktiven Verknüpfungen neu eingerichtet werden müssen oder wenn der Benutzer während der Arbeit mit einem eingebetteten Objekt den Befehl *Speichern unter* aus dem Menü *Datei* wählt.

Server-Anwendungen sollten geöffnete Dokumente für die Server-Bibliothek registrieren und diese benachrichtigen, wenn ein Dokument umbenannt, geschlossen, gespeichert oder in einem geänderten Zustand wiederhergestellt wird.

Siehe auch

OleRegisterServerDoc, *OleRevertServerDoc*, *OleRevokeServerDoc*, *OleSavedServerDoc*

OleRequestData 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleRequestData(lpObject, cfFormat)

LPOLEOBJECT lpObject;         /* Adresse des abzu-
                                 fragenden Objekts    */
OLECLIPFORMAT cfFormat;       /* Datenformat          */
```

Beschreibung

Die Funktion *OleRequestData* fordert die Bibliothek auf, Daten in einem bestimmten Format von einem Server abzurufen.

Parameter

lpObject: Zeigt auf das Objekt, das dem Server zugeordnet ist, von dem Daten angefordert werden sollen.

cfFormat: Bestimmt das Format, in dem die Daten zurückgegeben werden sollen. Dieser Parameter kann eines der vordefinierten Zwischenablageformate sein oder der von der Funktion *RegisterClipboardFormat* zurückgelieferte Wert.

Rückgabewert

Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_BUSY
OLE_ERROR_NOT_OPEN
OLE_ERROR_OBJECT
OLE_ERROR_STATIC
OLE_WAIT_FOR_RELEASE

Hinweis Die Client-Anwendung sollte mit dem Server verbunden sein, wenn sie die Funktion *OleRequestData* aufruft. Wenn der Client die Botschaft *OLE_RELEASE* erhält, kann er die Daten mit Hilfe der Funktion *OleGetData* von dem Objekt abrufen oder die Daten abfragen, indem er Funktionen wie z.B. *OleQueryBounds* verwendet.

Wenn die angeforderten Daten das gleiche Format wie die Präsentationsdaten für das Objekt haben, verwaltet die Bibliothek die Daten und aktualisiert die Präsentation.

Wenn der Server beschäftigt ist, gibt *OleRequestData OLE_WAIT_FOR_RELEASE* zurück, In diesem Fall sollte die Anwendung solange mit dem Senden von Botschaften fortfahren, bis sie eine Callback-Benachrichtigung mit dem Argument *OLE_RELEASE* erhält.

Siehe auch *OleEnumFormats, OleGetData, OleSetData, RegisterClipboardformat*

OleRevertClientDoc 3.1

Syntax
```
#include <ole.h>
OLESTATUS OleRevertClientDoc(lhClientDoc)

LHCLIENTDOC lhClientDoc;       /* Handle des
                                  Client-Dokuments    */
```

Beschreibung Die Funktion *OleRevertClientDoc* informiert die Bibliothek, daß ein Dokument in einem vorher gespeicherten Zustand wiederhergestellt worden ist.

Parameter *lhClientDoc:* Bezeichnet das Dokument, das in seinem gespeicherten Zustand wiederhergestellt worden ist.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann der Fehlerwert *OLE_ERROR_HANDLE* zurückgegeben werden.

Hinweis Eine Client-Anwendung sollte die Funktion *OleRevertClientDoc* aufrufen, wenn sie ein Dokument erneut lädt, ohne die letzten Änderungen gespeichert zu haben.

Client-Anwendungen sollten geöffnete Dokumente für die Bibliothek registrieren und sie benachrichtigen, wenn ein Dokument umbenannt, geschlossen, gespeichert oder in einem geänderten Zustand wiederhergestellt wird.

Siehe auch *OleRegisterClientDoc, OleRenameClientDoc, OleRevokeClientDoc, OleSavedClientDoc*

OleRevertServerDoc 3.1

Syntax
```
#include <ole.h>
OLESTATUS OleRevertServerDoc(lhDoc)
```

```
LHSERVERDOC lhDoc;       /* Handle des Dokuments      */
```

Beschreibung Die Funktion *OleRevertServerDoc* informiert die Server-Bibliothek, daß der Server ein Dokument in seinem gespeicherten Zustand wiederhergestellt hat, ohne es zu schließen.

Parameter *lhDoc:* Bezeichnet das Dokument, das in seinem gespeicherten Zustand wiederhergestellt worden ist.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann der Fehlerwert *OLE_ERROR_HANDLE* zurückgegeben werden.

Hinweis Server-Anwendungen sollten geöffnete Dokumente für die Bibliothek registrieren und sie benachrichtigen, wenn ein Dokument umbenannt, geschlossen, gespeichert oder in einem geänderten Zustand wiederhergestellt wird.

Siehe auch *OleRegisterServerDoc*, *OleRenameServerDoc*, *OleRevokeServerDoc*, *OleSavedServerDoc*

OleRevokeClientDoc 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleRevokeClientDoc(lhClientDoc)

LHCLIENTDOC lhClientDoc; /* Handle des Client-
                            Dokuments              */
```

Beschreibung Die Funktion *OleRevokeClientDoc* informiert die Client-Bibliothek, daß ein Dokument nicht mehr geöffnet ist.

Parameter *lhClientDoc:* Bezeichnet das Dokument, das nicht mehr geöffnet ist. Dieses Handle ist nach dem Aufruf der Funktion *OleRevokeClientDoc* ungültig.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_HANDLE
OLE_ERROR_NOT_EMPTY

Hinweis Die Client-Anwendung sollte vor dem Aufruf der Funktion *OleRevokeClientDoc* alle Objekte im Dokument löschen.

Client-Anwendungen sollten geöffnete Dokumente für die Bibliothek registrieren und sie benachrichtigen, wenn ein Dokument umbenannt, geschlossen, gespeichert oder in einem geänderten Zustand wiederhergestellt wird.

Siehe auch *OleRegisterClientDoc*, *OleRenameClientDoc*, *OleRevertClientDoc*, *OleSavedClientDoc*

OleRevokeObject 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleRevokeObject(lpClient)

LPOLECLIENT lpClient;          /* Adresse der
                                  OLECLIENT-Struktur   */
```

Beschreibung Die Funktion *OleRevokeObject* hebt den Zugriff auf ein Objekt auf. Eine Server-Anwendung ruft diese Funktion normalerweise auf, wenn der Benutzer ein Objekt freigibt und entfernt.

Parameter *lpClient:* Zeigt auf die *OLECLIENT*-Struktur, die dem Objekt zugeordnet ist, das freigegeben und entfernt werden soll.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Siehe auch *OleRevokeServer*, *OleRevokeServerDoc*

OleRevokeServer 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleRevokeServer(lhServer)

LHSERVER lhServer;     /* Handle des Server          */
```

Beschreibung *OleRevokeServer* wird von einer Server-Anwendung aufgerufen, um alle registrierten Dokumente zu schließen.

Parameter *lhServer*: Bezeichnet den Server. Eine Server-Anwendung kann dieses Handle durch die Funktion *OleRegisterServer* erhalten.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_HANDLE
OLE_WAIT_FOR_RELEASE

Hinweis Die Funktion *OleRevokeServer* liefert *OLE_WAIT_FOR_RELEASE* zurück, wenn Kommunikationen zwischen Client-Anwendungen und dem Server beendet werden sollen. In diesem Fall sollte die Server-Anwendung das Senden von Botschaften solange fortsetzen, bis ihre *Release*-Funktion von der Bibliothek aufgerufen wird.

Siehe auch *OleRegisterServer*, *OleRevokeObject*, *OleRevokeServerDoc*

OleRevokeServerDoc 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleRevokeServerDoc(lhDoc)
```

```
LHSERVERDOC lhDoc;      /* Handle des Dokuments        */
```

Beschreibung Die Funktion *OleRevokeServerDoc* hebt den Zugriff auf das angegebene Dokument auf. Eine Server-Anwendung ruft diese Funktion auf, wenn ein registriertes Dokument geschlossen oder anderweitig für Client-Anwendungen unzugänglich gemacht wird.

Parameter *lhDoc:* Bezeichnet das Dokument, für das der Zugriff aufgehoben werden soll. Dieses Handle wurde von der Funktion *OleRegisterServerDoc* zurückgeliefert oder ist dem Dokument durch eine der Server-Funktionen zugeordnet worden, die Dokumente erzeugen.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_HANDLE
OLE_WAIT_FOR_RELEASE

Hinweis Wenn diese Funktion *OLE_WAIT_FOR_RELEASE* zurückgibt, sollte die Server-Anwendung die *OLESERVERDOC*-Struktur nicht freigeben oder beendet werden, bis ihre *Release*-Funktion von der Bibliothek aufgerufen wird.

Siehe auch *OleRegisterServerDoc*, *OleRevokeObject*, *OleRevokeServer*

OleSavedClientDoc 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleSavedClientDoc(lhClientDoc)

LHCLIENTDOC lhClientDoc;   /* Handle des
                              Client-Dokuments        */
```

Beschreibung Die Funktion *OleSavedClientDoc* informiert die Client-Bibliothek, daß ein Dokument gespeichert worden ist.

Parameter *lhClientDoc:* Bezeichnet das Dokument, das gespeichert worden ist.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann der Fehlerwert *OLE_ERROR_HANDLE* zurückgegeben werden.

Hinweis Client-Anwendungen sollten geöffnete Dokumente für die Client-Bibliothek registrieren und sie benachrichtigen, wenn ein Dokument umbenannt, geschlossen, gespeichert oder in einem geänderten Zustand wiederhergestellt wird.

Siehe auch *OleRegisterClientDoc*, *OleRenameClientDoc*, *OleRevertClientDoc*, *OleRevokeClientDoc*

OleSavedServerDoc 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleSavedServerDoc(lhDoc)

LHSERVERDOC lhDoc;      /* Handle des Dokuments      */
```

Beschreibung Die Funktion *OleSavedServerDoc* informiert die Server-Bibliothek, daß ein Dokument gespeichert worden ist.

Parameter *lhDoc:* Bezeichnet das Dokument, das gespeichert worden ist.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_CANT_UPDATE_CLIENT
OLE_ERROR_HANDLE

Hinweis Ein Aufruf der Funktion *OleRenameServerDoc* entspricht dem Senden der Benachrichtigung *OLE_SAVED* an die Callback-Funktion der Client-Anwendung. Die Server-Anwendung ruft diese Funktion auf, wenn sie ein Dokument speichert oder wenn sie ein eingebettetes Objekt aktualisiert, ohne das Dokument zu schließen.

Empfängt eine Server-Anwendung den Fehlerwert *OLE_ERROR_CANT_UPDATE_CLIENT*, sollte sie ein Fenster mit der Meldung anzeigen, daß der Benutzer das Dokument erst aktualisieren kann, wenn der Server beendet ist.

Server-Anwendungen sollten geöffnete Dokumente für die Server-Bibliothek registrieren und sie benachrichtigen, wenn ein Dokument umbenannt, geschlossen, gespeichert oder in einem geänderten Zustand wiederhergestellt wird.

Siehe auch *OleRegisterServerDoc*, *OleRenameServerDoc*, *OleRevertServerDoc*, *OleRevokeServerDoc*

OleSaveToStream 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleSaveToStream(lpObject, lpStream)

LPOLEOBJECT lpObject; /* Adresse des zu speichernden
                         Objekts                    */
LPOLESTREAM lpStream; /* Adresse der OLESTREAM-
                         Struktur                   */
```

Beschreibung Die Funktion *OleSaveToStream* speichert ein Objekt in den Stream.

Parameter *lpObject:* Zeigt auf das Objekt, das in den Stream gespeichert werden soll.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_BLANK
OLE_ERROR_MEMORY
OLE_ERROR_OBJECT
OLE_ERROR_STREAM

Hinweis Eine Anwendung kann mit Hilfe der Funktion *OleQuerySize* ermitteln, wieviel Byte für das Objekt reserviert werden müssen.

Siehe auch *OleLoadFromStream*, *OleQuerySize*

OleSetBounds 3.1

Syntax
```
#include <ole.h>
OLESTATUS OleSetBounds(lpObject, lprcBound)

LPOLEOBJECT lpObject;     /* Adresse des Objekts      */
RECT FAR* lprcBound;      /* Adresse der Struktur     */
```

Beschreibung Die Funktion *OleSetBounds* bestimmt die Koordinaten des umgebenden Rechtecks für das angegebene Objekt auf dem Zielgerät.

Parameter *lpObjecT:* Zeigt auf das Objekt, für das das umgebende Rechteck festgelegt wird.

lprcBound: Zeigt auf eine *RECT*-Struktur, die die Koordinaten des Rechtecks enthält. Die Koordinaten werden in *MM_HIMETRIC*-Einheiten angegeben. Weder die Breite, noch die Höhe des Rechtecks sollte 32767 *MM_HIMETRIC*-Einheiten überschreiten.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_BUSY
OLE_ERROR_MEMORY
OLE_ERROR_OBJECT
OLE_WAIT_FOR_RELEASE

Die Funktion *OleSetBounds* gibt *OLE_ERROR_OBJECT* zurück, wenn sie für ein verknüpftes Objekt aufgerufen wird.

Hinweis Die Funktion *OleSetBounds* wird bei verknüpften Objekten ignoriert, weil deren Größe vom Quelldokument der Verknüpfung bestimmt wird.

Eine Client-Anwendung kann durch die Funktion *OleSetBounds* das umgebende Rechteck ändern. Sie muß nicht

jedesmal diese Funktion aufrufen, wenn ein Server geöffnet wird.

Das in der Funktion *OleSetBounds* festgelegte umgebende Rechteck hat nicht notwendigerweise die gleichen Abmessungen, wie das in einem Aufruf der Funktion *OleDraw* angegebene Rechteck. Die Abmessungen können wegen der von der Client-Anwendung verwendeten Bildschirmskalierung unterschiedlich sein. Eine Anwendung kann den Server mit Hilfe der Funktion *OleSetBounds* veranlassen, das Format des Bilds dem Rechteck anzupassen.

Im Abbildungsmodus *MM_HIMETRIC* verläuft die positive y-Richtung nach oben.

Siehe auch *OleDraw*, *OleQueryBounds*, *SetMapMode*

OleSetColorScheme 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleSetColorScheme(lpObject, lpPalette);

LPOLEOBJECT lpObject;                /* Adresse des
                                        Objekts      */
OLE_CONST LOGPALETTE FAR* lpPalette  /* Adresse der
                                        Palette      */
```

Beschreibung Die Funktion *OleSetColorScheme* legt die von einer Client-Anwendung empfohlene Palette fest, wenn die Server-Anwendung das angegebene Objekt bearbeitet. Die Server-Anwendung kann diese Empfehlung ignorieren.

Parameter *lpObject:* Zeigt auf eine *OLEOBJECT*-Struktur, die das Objekt beschreibt, für das die Client-Anwendung eine Palette vorschlägt.

lpPalette: Zeigt auf eine *LOGPALETTE*-Struktur, die die vorgeschlagene Palette angibt.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_BUSY
OLE_ERROR_COMM
OLE_ERROR_MEMORY
OLE_ERROR_OBJECT
OLE_ERROR_PALETTE
OLE_ERROR_STATIC
OLE_WAIT_FOR_RELEASE

Die Funktion *OleSetColorScheme* gibt *OLE_ERROR_OBJECT* zurück, wenn sie für ein verknüpftes Objekt aufgerufen wird.

Hinweis

Eine Client-Anwendung kann mit Hilfe der Funktion *OleSetColorScheme* das Farbschema ändern. Sie muß diese Funktion nicht jedesmal aufrufen, wenn ein Server geöffnet wird.

Der erste Paletteneintrag in der *LOGPALETTE*-Struktur legt die von der Client-Anwendung vorgeschlagene Vordergrundfarbe fest. Der zweite bestimmt die Hintergrundfarbe. Die erste Hälfte der restlichen Paletteneinträge sind Füllfarben und die zweite Hälfte Farben für Linien und Text.

Client-Anwendungen geben normalerweise eine gerade Anzahl von Paletteneinträgen an. Bei einer ungeraden Anzahl sollte der Server den ungeraden Eintrag als Füllfarbe betrachten. Wenn es z.B. fünf Einträge gibt, sollten drei als Füllfarben und zwei als Farben für Linien und Text interpretiert werden.

Server-Anwendungen sollten bei der Ausgabe von Metafiles die vorgeschlagene Palette verwenden.

OleSetData 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleSetData(lpObject, cfFormat, hData)

LPOLEOBJECT lpObject;      /* Adresse des Objekts     */
OLECLIPFORMAT cfFormat;    /* Datenformat             */
HANDLE hData;              /* Speicher, der die
                              Daten enthält           */
```

Beschreibung

Die Funktion *OleSetData* sendet Daten im angegebenen Format an den Server, der dem angegebenen Objekt zugeordnet ist.

Parameter

lpObject: Zeigt auf ein Objekt. Dadurch wird der Server bestimmt, an den Daten gesendet werden sollen.

cfFormat: Bestimmt das Format der Daten.

hData: Bezeichnet ein Speicherobjekt, das die Daten im angegebenen Format enthält.

Rückgabewert

Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_BUSY
OLE_ERROR_BLANK
OLE_ERROR_MEMORY
OLE_ERROR_NOT_OPEN
OLE_ERROR_OBJECT
OLE_WAIT_FOR_RELEASE

Wenn das Objekt die Daten nicht akzeptieren kann, gibt die Funktion einen Fehlerwert zurück. Wenn der Server nicht geöffnet ist und das Format der Daten nicht mit dem der

Präsentationsdaten übereinstimmt, wird der Wert *OLE_-ERROR_NOT_OPEN* zurückgeliefert.

Siehe auch *OleGetData*, *OleRequestData*

OleSetHostNames 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleSetHostNames(lpObject, lpszClient,
   lpszClientObj)

LPOLEOBJECT lpObject;    /* Adresse des Objekts       */
LPCSTR lpszClient;       /* Adresse des Namen-String
                            der Client-Anwendung      */
LPCSTR lpszClientObj;    /* Adresse des Objektnamen-
                            String                    */
```

Beschreibung Die Funktion *OleSetHostNames* legt den Namen der Client-Anwendung und den Namen des Client für das angegebene Objekt fest. Diese Informationen werden in Fenstertiteln verwendet, wenn das Objekt in der Server-Anwendung bearbeitet wird.

Parameter *lpObject:* Zeigt auf das Objekt, für das ein Name festgelegt werden soll.

lpszClient: Zeigt auf einen nullterminierten String, der den Namen der Client-Anwendung enthält.

lpszClientObj: Zeigt auf einen nullterminierten String, der den Namen der Client-Anwendung für das Objekt angibt.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_BUSY
OLE_ERROR_MEMORY
OLE_ERROR_OBJECT
OLE_WAIT_FOR_RELEASE

Die Funktion *OleSetHostNames* gibt *OLE_ERROR_OBJECT* zurück, wenn sie für ein verknüpftes Objekt aufgerufen wird.

Hinweis Wenn eine Server-Anwendung zum Bearbeiten eines eingebetteten Objekts gestartet wird, zeigt sie in der Titelzeile den String an, auf den der Parameter *lpszClientObj* zeigt. Dieser Objektname sollte der Name des Client-Dokuments sein, das das Objekt enthält.

Eine Client-Anwendung kann durch *OleSetHostNames* den Namen des Objekts festlegen, wenn es zum ersten Mal aktiviert wird oder dessen Namen ändern. Der Client muß diese Funktion nicht jedesmal aufrufen, wenn ein Server geöffnet wird.

OleSetLinkUpdateOptions 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleSetLinkUpdateOptions(lpObject, UpdateOpt)

LPOLEOBJECT lpObject;      /* Adresse des
                              verknüpften Objekts     */
OLEOPT_UPDATE UpdateOpt;   /* Aktualisierungsoptionen */
```

Beschreibung *OleSetLinkUpdateOptions* legt die Aktualisierungsoptionen für die Präsentation des angegebenen verknüpften Objekts fest.

Parameter *lpObject:* Zeigt auf das verknüpfte Objekt, für das die Aktualisierungsoptionen festgelegt werden.

UpdateOpt: Bestimmt die Aktualisierungsoptionen für das verknüpfte Objekt. Dieser Parameter kann einen der folgenden Werte annehmen:

Option	Beschreibung
oleupdate_always	Aktualisiert das verknüpfte Objekt sooft wie möglich. Diese Option unterstützt das Schaltfeld *Automatisch* im Dialogfenster *Verknüpfungen*.
oleupdate_oncall	Aktualisiert das verknüpfte Objekt nur auf Anforderung der Client-Anwendung. Diese Option unterstützt das Schaltfeld *Manuell* im Dialogfenster *Verknüpfungen*.
oleupdate_onsave	Aktualisiert das verknüpfte Objekt nur, wenn das Quelldokument von der Server-Anwendung gespeichert wird.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_BUSY
OLE_ERROR_OBJECT
OLE_ERROR_OPTION
OLE_ERROR_STATIC
OLE_WAIT_FOR_RELEASE

Siehe auch *OleGetLinkUpdateOptions*

OleSetTargetDevice 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleSetTargetDevice(lpObject, hotd)

LPOLEOBJECT lpObject;  /* Adresse des Objekts        */
```

```
HGLOBAL hotd;          /* Handle der OLETARGETDEVICE-
                          Struktur                    */
```

Beschreibung Die Funktion *OleSetTargetDevice* bestimmt das Ausgabegerät für ein Objekt.

Parameter *lpObject:* Zeigt auf das Objekt, für das ein Zielgerät festgelegt wird.

hotd: Bezeichnet eine *OLETARGETDEVICE*-Struktur, die das Zielgerät für das Objekt beschreibt.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_BUSY
OLE_ERROR_MEMORY
OLE_ERROR_OBJECT
OLE_ERROR_STATIC
OLE_WAIT_FOR_RELEASE

Hinweis Durch die Funktion *OleSetTargetDevice* kann ein verknüpftes oder eingebettetes Objekt einem bestimmten Zielgerät entsprechend formatiert werden, auch wenn es auf einem anderen Gerät ausgegeben wird. Eine Client-Anwendung sollte diese Funktion bei jeder Änderung des Zielgeräts aufrufen, damit Server-Anwendungen benachrichtigt werden können, nötigenfalls das Format des Objekts zu ändern. Die Client-Anwendung sollte dann die Funktion *OleUpdate* aufrufen, um sicherzustellen, daß die Informationen an den Server gesendet werden. So kann der Server die notwendigen Anpassungen der Präsentation des Objekts vornehmen. Danach sollte der Client die Bibliothek aufrufen, um das Objekt neu zu zeichnen, wenn er eine Benachrichtigung vom Server erhält, daß das Objekt geändert wurde.

Eine Client-Anwendung verwendet die Funktion *OleSetTargetDevice*, um das Zielgerät zu ändern. Sie muß diese Funktion nicht jedesmal aufrufen, wenn ein Server geöffnet wird.

OleUnblockServer 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleUnblockServer(lhSrvr, lpfRequest)

LHSERVER lhSrvr;        /* Handle des Server         */
BOOL FAR* lpfRequest;   /* Adresse des Flag, das
                           weitere Aufträge angibt   */
```

Beschreibung Die Funktion *OleUnblockServer* bearbeitet Aufträge in einer Warteschlange, die durch einen Aufruf der Funktion *OleBlockServer* erzeugt wurde.

Parameter *lhSrvr:* Bezeichnet den Server, für den Aufträge in die Warteschlange gestellt worden sind.

lpfRequest: Zeigt auf ein Flag, das angibt, ob weitere Aufträge in der Warteschlange vorhanden sind. Wenn dies der Fall ist, hat dieses Flag nach Beendigung der Funktion den Wert *TRUE*, andernfalls den Wert *FALSE*.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_HANDLE
OLE_ERROR_MEMORY

Hinweis Eine Server-Anwendung kann die Funktionen *OleBlockServer* und *OleUnblockServer* verwenden, um zu steuern, wann die Server-Bibliothek Anforderungen von Client-Anwendungen bearbeiten soll. *OleUnblockServer* sollte am besten außerhalb der Funktion *GetMessage* in einer Botschaftsschleife aufgerufen werden, um die Blockierung aller Botschaften aufzuheben, bevor die nächste Botschaft bearbeitet wird. Die Aufhebung der Blockierung von Botschaften sollte nicht innerhalb von server-definierten Funktionen erfolgen, die von der Bibliothek aufgerufen werden.

Siehe auch *OleBlockServer*

OleUnlockServer 3.1

Syntax
```
#include <ole.h>
OLESTATUS OleUnlockServer(hServer)

LHSERVER hServer;    /* Handle des Server, der
                        entsperrt werden soll        */
```

Beschreibung Die Funktion *OleUnlockServer* macht die Sperrung eines Server rückgängig, der vorher durch einen Aufruf der Funktion *OleLockServer* gesperrt wurde.

Parameter *lhServer:* Bezeichnet die Server-Anwendung, die aus dem Speicher entfernt werden soll. Dieses Handle wurde durch einen Aufruf der *OleLockServer* ermittelt.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_HANDLE
OLE_WAIT_FOR_RELEASE

Hinweis Wenn die Funktion *OleLockServer* mehr als einmal für einen angegebenen Server aufgerufen wird, wird der Sperrenzähler

des Server erhöht. Jeder Aufruf der Funktion *OleUnlockServer* verringert diesen Wert. Der Server bleibt gesperrt, bis sein Sperrenzähler den Wert Null erreicht hat.

Wenn die Funktion *OleUnlockServer OLE_WAIT_FOR_RELEASE* zurückgibt, sollte die Anwendung die Funktion *OleQueryReleaseStatus* aufrufen, um zu ermitteln, ob die Entsperrung abgeschlossen ist. Im Aufruf der Funktion *OleQueryReleaseStatus* kann die Anwendung den Typ des Server-Handle in einen **long**-Zeiger auf ein OLE-Objekt umwandeln (*LPOLEOBJECT*):

```
OleQueryReleaseStatus((LPOLEOBJECT) lhServer);
```

Wenn *OleQueryReleaseStatus* nicht mehr den Wert *OLE_BUSY* zurückgibt, ist der Server entsperrt worden.

Siehe auch *OleLockServer, OleQueryReleaseStatus*

OleUpdate 3.1

Syntax

```
#include <ole.h>
OLESTATUS OleUpdate(lpObject)

LPOLEOBJECT lpObject;    /* Adresse des Objekts       */
```

Beschreibung Die Funktion *OleUpdate* aktualisiert das angegebene Objekt. Sie aktualisiert die Präsentation des Objekts und stellt sicher, daß es in Bezug auf alle verknüpften Objekte, die es enthält, auf dem neuesten Stand ist.

Parameter *lpObject:* Zeigt auf das Objekt, das aktualisiert werden soll.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_BUSY
OLE_ERROR_OBJECT
OLE_ERROR_STATIC
OLE_WAIT_FOR_RELEASE

Siehe auch *OleQueryOutOfDate*

OpenDriver 3.1

Syntax

```
HDRVR OpenDriver(lpDriverName, lpSectionName, lParam)

LPCSTR lpDriverName;  /* Adresse des Treibernamens    */
LPCSTR lpSectionName; /* Adresse des Abschnittnamens
                         der .INI-Datei               */
LPARAM lParam;        /* Adresse der treiberspezifischen
                         Informationen                */
```

Beschreibung Die Funktion *OpenDriver* führt notwendige Initialisierungen durch, indem sie z.B. die Elemente in den Strukturen installierbarer Treiber auf ihre voreingestellten Werte setzt.

Parameter *lpDriverName:* Zeigt auf einen nullterminierten String, der den Namen eines installierbaren Treibers angibt.

lpSectionName: Zeigt auf einen nullterminierten String, der den Namen eines Abschnitts in der Datei SYSTEM.INI angibt.

lParam: Bestimmt treiberspezifische Informationen.

Rückgabewert Der Rückgabewert ist das Handle des installierbaren Treibers, wenn die Funktion erfolgreich ist. Andernfalls ist der Rückgabewert *NULL*.

Hinweis Der String, auf den *lpDriverName* zeigt, muß mit dem Namen des installierbaren Treibers in der Datei SYSTEM.INI identisch sein.

Wenn sich der Name des installierbaren Treibers im Abschnitt [drivers] der Datei SYSTEM.INI befindet, sollte der durch *lpSectionName* bezeichnete String *NULL* sein. Andernfalls sollte dieser String den Namen des Abschnitts in der Datei SYSTEM.INI angeben, der den Treibernamen enthält.

Öffnet eine Anwendung einen Treiber zum ersten Mal, ruft Windows die Funktion *DriverProc* mit den Botschaften *DRV_LOAD*, *DRV_ENABLE* und *DRV_OPEN* auf. Beim Öffnen untergeordneter Instanzen dieses Treibers wird nur *DRV_OPEN* übergeben.

Der im Parameter *lParam* angegebene Wert wird im Parameter *lParam2* der Funktion *DriverProc* übergeben.

Siehe auch *CloseDriver, Driver Proc*

PrintDlg

Syntax

```
BOOL PrintDlg(lppd)

LPRINTDLG lppd;          /* Zeiger auf eine Struktur
                            mit Initialisierungs-
                            daten                  */
```

Beschreibung Die Funktion *PrintDlg* zeigt ein *Drucken-* oder *Druckereinrichtung*-Dialogfenster an. Im *Drucken*-Dialogfenster kann der Anwender die Eigenschaften eines einzelnen Druckauftrags angeben.

Das Dialogfenster *Druckereinrichtung* ermöglicht es dem Anwender, zusätzliche Eigenschaften für den Druck zu wählen und den Drucker zu konfigurieren.

Parameter *lppd:* Zeigt auf eine *PRINTDLG*-Struktur, die Informationen für die Initialisierung des Dialogfensters enthält. Nach Rückkehr

der Funktion *PrintDlg* enthält diese Struktur Informationen über die Druckoptionen, die der Anwender gewählt hat.

Die *PRINTDLG*-Struktur hat folgende Form:

```
typedef struct tagPD  {   /* pd */
   DWORD lStructSize;
   HWND hwndOwner;
   HGLOBAL hDevMode;
   HGLOBAL hDevNames;
   HDC hDC;
   DWORD Flags;
   UINT nFromPage;
   UINT nToPage;
   UINT nMinPage;
   UINT nMaxPage;
   UINT nCopies;
   HINSTANCE hInstance;
   LPARAM lCustData;
   UINT (CALLBACK* lpfnPrintHook) (HWND, UINT,
      WPARAM, LPARAM);
   UINT (CALLBACK* lpfnSetupHook) (HWND, UINT,
      WPARAM, LPARAM);
   LPCSTR lpPrintTemplateName;
   LPCSTR lpSetupTemplateName;
   HGLOBAL hPrintTemplate;
   HGLOBAL hSetupTemplate;
} PRINTDLG;
```

Rückgabewert

Der Rückgabewert ist ungleich Null, wenn die Funktion den Drucker erfolgreich konfiguriert hat. Der Rückgabewert ist Null, wenn der Anwender den Schalter *Abbrechen* oder die Option *Schließen* aus dem Systemmenü gewählt hat. Der Rückgabewert ist auch Null, wenn der Anwender den Schalter *Einrichten* wählt, das Dialogfenster *Druckereinrichtung* mit dem *OK*-Schalter schließt und dann im Dialogfenster *Drucken* den Schalter *Abbrechen* wählt.

Fehler

Verwenden Sie die Funktion *CommDlgExtendedError*, um den Fehlerwert abzurufen. Möglich sind folgende Werte:

CDERR_FINDRESFAILURE
CDERR_INITIALISATION
CDERR_LOADRESFAILURE
CDERR_LOADSTRFAILURE
CDERR_LOCKRESFAILURE
CDERR_MEMALLOCFAILURE
CDERR_MEMLOCKFAILURE
CDERR_NOHINSTANCE
CDERR_NOHOOK
CDERR_NOTEMPLATE

CDERR_STRUCTSIZE
PDERR_CREATEICFAILURE
PDERR_DEFAULTDIFFERENT
PDERR_DNDMMISMATCH
PDERR_GETDEVMODEFAIL
PDERR_INITFAILURE
PDERR_LOADDRVFAILURE
PDERR_NODEFAULTPRN
PDERR_NODEVICES
PDERR_PARSEFAILURE
PDERR_PRINTERNOTFOUND
PDERR_RETDEFFAILURE
PDERR_SETUPFAILURE

Beispiel Das folgende Beispiel initialisiert die *PRINTDLG*-Struktur, zeigt das Dialogfenster *Drucken* mit Hilfe der Funktion *PrintDlg* an und druckt eine Beispielseite, wenn der Rückgabewert ungleich Null ist:

```
PRINTDLG pd;

/* Alle Elemente der Struktur mit Null
   initialisieren. */

memset (&pd, 0, sizeof(PRINTDLG));

/* Die benötigten Elemente der PRINTDLG-Struktur
   werden initialisiert. */

pd.lStructSize = sizeof(PRINTDLG);
pd.hwndOwner = hwnd;
pd.Flags = PD_RETURNDC;

/* Bei erfolgreicher Ausführung wird eine Testseite
   gedruckt. */

if (PrintDlg(&pd) != 0) {
    Escape (pd.hDC, STARTDOC, 8, "Test-Doc", NULL);

    /* Text und Rechteck drucken */

    TextOut(pd.hDC, 50, 50, "Common Dialog Test Page",
       23);
    Rectangle(pd.hDC, 50, 90, 625, 105);
    Escape(pd.hDC, NEWFRAME, 0, NULL, NULL);
    Escape(pd.hDC, ENDDOC, 0, NULL, NULL);
    DeleteDC(pd.hDC);
    if (pd.hDevMode != NULL)
       GlobalFree(pd.hDevMode);
    if (pd.hDevNames != NULL)
       GlobalFree(pd.hDevNames);
}
```

```
else
    ErrorHandler();
```

QueryAbort 3.1

Syntax

```
BOOL QueryAbort(hdc, reserved)

HDC hdc;       /* Handle des Gerätekontexts          */
int reserved;  /* reserviert, sollte Null sein       */
```

Beschreibung Die Funktion *QueryAbort* ruft die Callback-Funktion *AbortProc* für eine Druckanwendung auf und überprüft, ob der Ausdruck abgebrochen werden soll.

Parameter *hdc:* Bezeichnet den Gerätekontext.

message: Gibt einen reservierten Wert an. Sollte auf Null gesetzt werden.

Rückgabewert Der Rückgabewert ist *TRUE*, wenn der Ausdruck fortgesetzt werden soll oder keine Abbruchfunktion existiert. Er ist *FALSE*, wenn der Druckauftrag abgebrochen werden soll. Der Rückgabewert wird von der Callback-Funktion *AbortProc* bereitgestellt.

Siehe auch *AbortDoc, AbortProc, SetAbortProc*

QuerySendMessage 3.1

Syntax

```
BOOL QuerySendMessage(hreserved1, hreserved2,
   hreserved3, lpMessage)

HANDLE hreserved1;   /* Reserviert, muß NULL sein     */
HANDLE hreserved2;   /* Reserviert, muß NULL sein     */
HANDLE hreserved3;   /* Reserviert, muß NULL sein     */
LPMSG lpMessage;     /* Adresse der Struktur für die
                        Botschaft                     */
```

Beschreibung Die Funktion *QuerySendMessage* überprüft, ob eine mit der Funktion *SendMessage* übergebene Botschaft aus der aktuellen Task stammt. Handelt es sich um die Botschaft einer anderen Task, stellt *QuerySendMessage* diese Botschaft in die angegebene *MSG*-Struktur.

Parameter *hreserved1:* Reserviert, muß *NULL* sein.

hreserved2: Reserviert, muß *NULL* sein.

hreserved3: Reserviert, muß *NULL* sein.

lpMessage: Bestimmt die *MSG*-Struktur, in der die Botschaft abgelegt werden soll. Die *MSG*-Struktur hat folgende Form:

```
typedef struct tagMSG {    /* msg */
    HWND    hwnd;
    UINT    message;
    WPARAM  wParam;
    LPARAM  lParam;
    DWORD   time;
    POINT   pt;
} MSG;
```

Rückgabewert Der Rückgabewert ist Null, wenn die Botschaft aus der aktuellen Task stammt. Andernfalls ist er ungleich Null.

Hinweis Wenn der Windows-Debugger den Soft-Modus aktiviert, sollte die getestete Anwendung Botschaften anderer Tasks mit Hilfe der Funktion *ReplyMessage* beantworten.

Die Parameter mit dem Wert *NULL* sind für zukünftige Nutzungen reserviert.

Siehe auch *SendMessage, ReplyMessage*

RedrawWindow 3.1

Syntax

```
BOOL RedrawWindow(hwnd, lprcUpdate, hrgnUpdate,
   fuRedraw)

HWND hwnd;                 /* Handle des Fensters     */
const RECT FAR* lprcUpdate /* Adresse der Struktur
                              mit dem
                              Aktualisierungsrechteck */
HRGN hrgnUpdate;           /* Handle der
                              Aktualisierungsregion   */
UINT fuRedraw;             /* Aktualisierungs-Flag    */
```

Beschreibung Die Funktion *RedrawWindow* aktualisiert das angegebene Rechteck oder die angegebene Region im Client-Bereich des angegebenen Fensters.

Parameter *hwnd:* Bezeichnet das Fenster, das aktualisiert werden soll. Wenn dieser Parameter *NULL* ist, wird der Desktop aktualisiert.

lprcUpdate: Zeigt auf eine *RECT*-Struktur, die die Koordinaten des Aktualisierungsrechtecks enthält. Dieser Parameter wird ignoriert, wenn der Parameter *hrgnUpdate* ein gültiges Handle einer Region enthält. Die *RECT*-Struktur hat folgende Form:

```
typedef struct tagRECT {   /* rc */
    int left;
    int top;
    int right;
    int bottom;
} RECT;
```

hrgnUpdate: Bezeichnet die Aktualisierungsregion. Haben die beiden Parameter *hrgnUpdate* und *lprcUpdate* den Wert *NULL*, wird der gesamte Client-Bereich zur Aktualisierungsregion hinzugefügt.

fuRedraw: Bestimmt ein oder mehrere Aktualisierungs-Flags. Dieser Parameter kann eine Kombination der in den beiden folgenden Tabellen aufgeführten Flags sein. Diese Flags werden benutzt, um das Fenster ungültig zu machen:

Wert	Bedeutung
RDW_ERASE	Übergibt eine *WM_ERASEBKGND*-Botschaft an das Fenster, wenn das Fenster neu gezeichnet wird. Das Flag *RDW_INVALIDATE* muß ebenfalls angegeben werden, da *RDW_ERASE* sonst keine Auswirkungen hat.
RDW_FRAME	Jeder Teil des Nicht-Client-Bereichs des Fensters, der sich mit der Aktualisierungsregion überschneidet, erhält eine *WM_NCPAINT*-Botschaft. Das Flag *RDW_INVALIDATE* muß ebenfalls angegeben werden, andernfalls hat *RDW_FRAME* keinerlei Auswirkungen. Die Botschaft *WM_NCPAINT* wird normalerweise während der Ausführung der Funktion *RedrawWindow* nicht übergeben, es sei denn, daß entweder *RDW_UPDATENOW* oder *RDW_ERASENOW* angegeben ist.
RDW_INTERNAL-PAINT	Übergibt eine *WM_PAINT*-Botschaft an das Fenster, auch wenn dieses Fenster keine ungültige Region enthält.
RDW_INVALIDATE	*lprcUpdate* oder *hrgnUpdate* werden ungültig. Nur einer der beiden Parameter darf ungleich *NULL* sein. Haben beide Parameter den Wert *NULL*, wird das gesamte Fenster ungültig.

Diese Flags werden verwendet, um das Fenster gültig zu machen:

Wert	Bedeutung
RDW_NOERASE	Unterdrückt anhängige *WM_ERASEBKGND*-Botschaften.
RDW_NOFRAME	Unterdrückt anhängige *WM_NCPAINT*-Botschaften. Dieses Flag muß zusammen mit *RDW_VALIDATE* verwendet werden. Meistens wird auch das Flag *RDW_NOCHILDREN* angegeben.

Wert	Bedeutung
	Diese Option sollte vorsichtig verwendet werden, da sie dazu führen kann, daß Teile eines Fensters nicht korrekt gezeichnet werden.
RDW_NOINTERNALPAINT	Unterdrückt anhängige interne *WM_PAINT*-Botschaften. Dieses Flag beeinflußt keine *WM_PAINT*-Botschaften, die von ungültigen Regionen ausgehen.
RDW_VALIDATE	Macht *lprcUpdate* oder *hrgnUpdate* gültig. Nur einer der beiden Parameter darf ungleich *NULL* sein. Haben beide Parameter den Wert *NULL*, wird das gesamte Fenster gültig. Dieses Flag beeinflußt interne *WM_PAINT*-Botschaften nicht.

Die folgenden Flags steuern den Zeitpunkt des Neuzeichnens. Die Funktion *RedrawWindow* führt keine Neuzeichnung durch, solange nicht eins dieser Bits angegeben wurde:

Wert	Bedeutung
RDW_ERASENOW	Wenn nötig, empfangen die durch die Flags *RDW_ALLCHILDREN* und *RDW_NOCHILDREN* bezeichneten Fenster *WM_NCPAINT*- und *WM_ERASEBKGND*-Botschaften, bevor die Funktion zurückkehrt. *WM_PAINT*-Botschaften werden zurückgestellt.
RDW_UPDATENOW	Wenn nötig, erhalten die durch die Flags *RDW_ALLCHILDREN* und *RDW_NOCHILDREN* bezeichneten Fenster *WM_NCPAINT*-, *WM_ERASEBKGND*- und *WM_PAINT*-Botschaften, bevor die Funktion zurückkehrt.

In der Voreinstellung bestimmt das Stil-Bit *WS_CLIPCHILDREN* des angegebenen Fensters, welche Fenster von der Funktion *RedrawWindow* beeinflußt werden. Die untergeordneten Fenster von Fenstern mit dem gesetzten Stil-Bit *WS_CLIPCHILDREN* werden nicht beeinflußt.

Die untergeordneten Fenster eines Fensters, bei dem das Stil-Bit *WS_CLIPCHILDREN* nicht gesetzt ist, werden rekursiv gültig bzw. ungültig gemacht, bis ein Fenster mit dem Bit *WS_CLIPCHILDREN* angetroffen wird.

Die folgenden Flags steuern, welche Fenster von der Funktion *RedrawWindow* beeinflußt werden:

Wert	Bedeutung
RDW_ALL-CHILDREN	Alle untergeordneten Fenster werden in die Aktualisierung eingeschlossen.
RDW_NO-CHILDREN	Untergeordnete Fenster werden von der Aktualisierung ausgeschlossen.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Wird die Funktion *RedrawWindow* benutzt, um Teile des Desktop ungültig zu machen, erhält das Desktop-Fenster keine *WM_PAINT*-Botschaft. Um den Desktop neu zu zeichnen, sollten Anwendungen das Flag *RDW_ERASE* benutzen, um eine *WM_ERASEBKGND*-Botschaft zu erzeugen.

Siehe auch *GetUpdateRect*, *GetUpdateRgn*, *InvalidateRect*, *InvalidateRgn*, *UpdateWindow*

RegCloseKey 3.1

Syntax
```
#include <shellapi.h>
LONG RegCloseKey(hkey)

HKEY hkey;   /* Handle des zu schließenden Schlüssels */
```

Beschreibung Die Funktion *RegCloseKey* schließt einen Schlüssel. Das Schließen eines Schlüssels gibt dessen Handle frei. Sobald alle Schlüssel geschlossen sind, wird die Registrierungsdatenbank aktualisiert.

Parameter *hkey:* Bezeichnet den zu schließenden Schlüssel.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert *ERROR_SUCCESS*. Andernfalls handelt es sich um einen Fehlerwert.

Hinweis Die Funktion *RegCloseKey* sollte nur aufgerufen werden, wenn ein Schlüssel mit der Funktion *RegOpenKey* oder *RegCreateKey* geöffnet wurde.

Das Handle eines angegebenen Schlüssels ist ungültig, nachdem der Schlüssel geschlossen wurde. Schlüssel-Handle sollten sofort geschlossen werden, wenn sie nicht mehr benötigt werden.

Beispiel Das folgende Beispiel benutzt die Funktion *RegCreateKey*, um das Handle eines Protokolls zu erzeugen. Anschließend setzt die Funktion *RegSetValue* die untergeordneten Schlüssel des Protokolls. Dann wird *RegCloseKey* aufgerufen, um die Informationen in der Datenbank zu speichern:

```
HKEY hkProtocol;

                /* Hauptschlüssel */
if (RegCreateKey(HKEY_CLASSES_ROOT,
                /* Protokollstring */
   "NewAppDocument\\protocol\\StdFileEditing",
                /* Handle des Protokollschlüssels */
   &hkProtocol) != ERROR_SUCCESS) return FALSE;

                /* Handle des Protokollschlüssels */
   RegSetValue(hkProtocol),
 "server",      /* Name des untergeordneten
                   Schlüssels */

   REG_SZ,      /* muß angegeben werden */
  "newapp.exe", /* Befehl zum Aktivieren
                   der Server-Anwendung */
   10);         /* Größe des Text-String */

                /* Handle des Protokollschlüssels */
   RegSetValue(hkProtocol,
   "verb\\0",   /* Name des untergeordneten
                   Schlüssels */
   REG_SZ,      /* muß angegeben werden */
   "EDIT",      /* Server-Anwendung sollte das Objekt
                   zum Bearbeiten bereitstellen */
   4);          /* Größe des Text-String */

                /* Schließt den Protokollschlüssel
                   und die untergeordneten
                   Schlüssel */
RegCloseKey(hkProtocol);
```

Siehe auch *RegCreateKey, RegDeleteKey, RegOpenKey, RegSetValue*

RegCreateKey 3.1

Syntax

```
#include <shellapi.h>
LONG RegCreateKey (hkey, lpszSubKey, lphkResult)

HKEY hkey;             /* Handle eines geöffneten
                          Schlüssels                */
LPCSTR lpszSubKey;     /* Adresse des String für den
                          zu öffnenden untergeordneten
                          Schlüssel                 */
HKEY FAR* lphkResult;  /* Adresse des Handle des
                          geöffneten Schlüssels     */
```

Beschreibung Die Funktion *RegCreateKey* erzeugt den angegebenen Schlüssel. Wenn der Schlüssel bereits in der Registrierungsdatenbank existiert, öffnet *RegCreateKey* ihn.

Parameter

hkey: Bezeichnet einen geöffneten Schlüssel. Dabei kann es sich um *HKEY_CLASSES_ROOT* handeln. Der von der Funktion *RegCreateKey* geöffnete oder erzeugte Schlüssel ist ein untergeordneter Schlüssel des durch den Parameter *hkey* bezeichneten Schlüssels. Dieser Wert sollte nicht *NULL* sein.

lpszSubKey: Zeigt auf einen nullterminierten String, der den zu öffnenden oder erzeugenden untergeordneten Schlüssel angibt.

lphkResult: Zeigt auf das Handle des Schlüssels, der geöffnet oder erzeugt werden soll.

Rückgabewert

Wenn die Funktion erfolgreich ist, ist der Rückgabewert *ERROR_SUCCESS*. Andernfalls handelt es sich um einen Fehlerwert.

Hinweis

Anwendungen können untergeordnete Schlüssel in der Datenbank erzeugen, indem sie *HKEY_CLASSES_ROOT* im Parameter *hkey* angeben. Anwendungen können die Funktion *RegCreateKey* benutzen, um mehrere Schlüssel gleichzeitig zu erzeugen. So kann eine Anwendung z. B. einen untergeordneten Schlüssel der vierten Ebene und die drei vorhergehenden untergeordneten Schlüssel erzeugen, indem sie einen String der folgenden Form im Parameter *lpszSubKey* angibt:

```
subkey1/subkey2/subkey3/subkey4
```

Beispiel

Das folgende Beispiel erzeugt das Handle eines Protokolls mit Hilfe der Funktion *RegCreateKey*, setzt dann die untergeordneten Schlüssel des Protokolls mit der Funktion *RegSetValue* und ruft anschließend *RegCloseKey* auf, um die Informationen in der Datenbank zu speichern:

```
if (RegCreateKey(HKEY_CLASSES_ROOT,
                            /* Hauptschlüssel */
    "NewAppDocument\\protocol\\StdFileEditing",
                            /* Protokollstring */
    &hkProtocol) != ERROR_SUCCESS)
                            /* Handle des Protokoll-
                               schlüssels */
         return FALSE;

RegSetValue(hkProtocol,    /* Handle des Protokoll-
                              schlüssels */
    "server",              /* Name des untergeordneten
                              Schlüssels */
    REG_SZ,                /* muß angegeben werden */
    "newapp.exe",          /* Befehl zum Aktivieren
                              der Server-Anwendung */
    10);                   /* Größe des Text-String */
```

```
RegSetValue(hkProtocol,    /* Handle des Protokoll-
                              schlüssels */
    "verb\\0",             /* Name des untergeordneten
                              Schlüssels */
    REG_SZ,                /* muß angegeben werden */
    "EDIT",                /* Server sollte das Objekt
                              zum Bearbeiten
                              bereitstellen */
    4);                    /* Größe des Text-String */

RegCloseKey(hkProtocol);   /* Schließt den Protokoll-
                              schlüssel und die unter-
                              geordneten Schlüssel */
```

Siehe auch *RegCloseKey, RegOpenKey, RegSetValue*

RegDeleteKey 3.1

Syntax

```
#include <shellapi.h>
LONG RegDeleteKey(hkey, lpszSubKey)

HKEY hkey;             /* Handle eines
                          geöffneten Schlüssels      */
LPCSTR lpszSubKey;     /* Adresse des String für den zu
                          löschenden untergeordneten
                          Schlüssel                  */
```

Beschreibung Die Funktion *RegDeleteKey* löscht den angegebenen Schlüssel. Wenn ein Schlüssel gelöscht wird, wird sein Wert und der Wert all seiner untergeordneten Schlüssel gelöscht.

Parameter *hkey:* Bezeichnet einen geöffneten Schlüssel. Dabei kann es sich um *HKEY_CLASSES_ROOT* handeln. Der durch die Funktion *RegDeleteKey* gelöschte Schlüssel ist ein untergeordneter Schlüssel des hier angegebenen Schlüssels.

lpszSubKey: Zeigt auf einen nullterminierten String, der den zu löschenden untergeordneten Schlüssel angibt. Dieser Wert sollte nicht *NULL* sein.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert *ERROR_SUCCESS*. Andernfalls handelt es sich um einen Fehlerwert.

Wenn der Fehlerwert *ERROR_ACCESS_DENIED* ist, besitzt die Anwendung entweder nicht die nötigen Rechte, um den angegebenen Schlüssel zu löschen oder eine andere Anwendung hat den angegebenen Schlüssel geöffnet.

Beispiel Das folgende Beispiel verwendet die Funktion *RegQueryValue*, um den Namen eines Objekt-Handler abzurufen. Anschließend ruft sie *RegDeleteKey*, um den Schlüssel zu löschen, wenn er den Wert »nwappobj.dll« hat:

```
char szBuff[80];
LONG cb;
HKEY hkStdFileEditing;

if (RegOpenKey(HKEY_CLASSES_ROOT,
        "NewAppDocument\\protocol\\StdFileEditing",
        &hkStdFileEditing) == ERROR_SUCCESS) {

    cb = sizeof(szBuff);

if (RegQueryValue(hkStdFileEditing,
            "handler",
            szBuff,
            &cb) == ERROR_SUCCESS
            && lstrcmpi("nwappobj.dll", szBuff) == 0)
        RegDeleteKey(hkStdFileEditing, "handler");
    RegCloseKey(hkStdFileEditing);
}
```

Siehe auch *RegCloseKey*

RegEnumKey 3.1

Syntax

```
#include <shellapi.h>
LONG RegEnumKey(hkey, iSubkey, lpszBuffer, cbBuffer)

HKEY hkey;        /* Handle des zu untersuchenden
                     Schlüssels                    */
DWORD iSubkey;    /* Index des zu untersuchenden
                     untergeordneten Schlüssels    */
LPSTR lpszBuffer; /* Adresse des Puffers für den String
                     des untergeordneten Schlüssels */
DWORD cbBuffer;   /* Größe des Puffers             */
```

Beschreibung Die Funktion *RegEnumKey* listet die untergeordneten Schlüssel eines angegebenen Schlüssels auf.

Parameter *hkey:* Bezeichnet einen geöffneten Schlüssel. Dabei kann es sich auch um *HKEY_CLASSES_ROOT* handeln. Für diesen Schlüssel werden Informationen über seine untergeordneten Schlüssel abgerufen.

iSubkey: Bestimmt den Index des abzurufenden untergeordneten Schlüssels. Dieser Wert sollte beim ersten Aufruf der Funktion *RegEnumKey* Null sein.

lpszBuffer: Zeigt auf einen Puffer, der bei der Rückkehr der Funktion den Namen des untergeordneten Schlüssels enthält. Die Funktion kopiert nur den Namen des untergeordneten Schlüssels, nicht die vollständige Hierarchie des Schlüssels in den Puffer.

cbBuffer: Bestimmt die Größe des durch den Paramter *lpszBuffer* bezeichneten Puffers in Byte.

Rückgabewert Ist die Funktion erfolgreich, ist der Rückgabewert *ERROR_SUCCESS*. Andernfalls handelt es sich um einen Fehlerwert.

Hinweis Der erste Parameter der Funktion *RegEnumKey* muß einen geöffneten Schlüssel angeben. Vor dem Aufruf der Funktion *RegEnumKey* erfolgt deshalb normalerweise ein Aufruf der Funktion *RegOpenKey* und anschließend ein Aufruf der Funktion *RegCloseKey*. Der Aufruf von *RegOpenKey* und *RegCloseKey* ist nicht notwendig, wenn der erste Parameter den Wert *HKEY_CLASSES_ROOT* hat, da dieser Schlüssel jederzeit geöffnet und verfügbar ist. Auch in diesem Fall führt der Aufruf von *RegOpenKey* und *RegCloseKey* allerdings zu einer Beschleunigung. Wenn eine Anwendung die Funktion *RegEnumKey* benutzt, sollte sie keine Aufrufe von Registrierfunktionen durchführen, die den abgefragten Schlüssel verändern könnten.

Um untergeordnete Schlüssel aufzulisten, sollten Anwendungen anfangs den Parameter *iSubkey* auf Null setzen und ihn anschließend bei weiteren Aufrufen erhöhen.

Beispiel Das folgende Beispiel benutzt die Funktion *RegEnumKey*, um die den Hauptschlüsseln zugeordnete Werte in eine Liste zu schreiben:

```
HKEY hkRoot;
char szBuff[80], szValue[80];
static DWORD dwIndex;
LONG cb;

if (RegOpenKey(HKEY_CLASSES_ROOT, NULL,
   &hkRoot) == ERROR_SUCCESS) {
      for (dwIndex = 0; RegEnumKey(hkRoot, dwIndex,
           szbuff, sizeof(szBuff)) == ERROR_SUCCESS;
           ++dwIndex) {
            if (*szBuff == '.')
                continue;
            cb = sizeof(szValue);
            if (RegQueryValue(hkRoot, (LPSTR) szBuff,
            szValue, &cb) == ERROR_SUCCESS
            SendDlgItemMessage(hDlg, ID_ENUMLIST,
              LB_ADDSTRING, 0, (LONG) (LPSTR) szValue);
    }
    RegCloseKey(hkRoot);
}
```

Siehe auch *RegQueryValue*

RegOpenKey 3.1

Syntax

```
#include <shellapi.h>
LONG RegOpenKey(hkey, lpszSubKey, lphkResult)

HKEY hkey;            /* Handle eines geöffneten
                         Schlüssels                  */
LPCSTR lpszSubKey;    /* Adresse des String des zu
                         öffnenden untergeordneten
                         Schlüssels                  */
HKEY FAR* lphkResult; /* Adresse des Handle des
                         geöffneten Schlüssels       */
```

Beschreibung Die Funktion *RegOpenKey* öffnet den angegebenen Schlüssel.

Parameter *hkey:* Bezeichnet einen geöffneten Schlüssel. Dabei kann es sich um *HKEY_CLASSES_ROOT* handeln. Der durch die Funktion *RegOpenKey* geöffnete Schlüssel ist ein untergeordneter Schlüssel des durch diesen Parameter bezeichneten Schlüssels. Dieser Wert darf nicht *NULL* sein.

lpszSubKey: Zeigt auf einen nullterminierten String, der den Namen des zu öffnenden untergeordneten Schlüssels angibt.

lphkResult: Zeigt auf das Handle des geöffneten Schlüssels.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert *ERROR_SUCCESS*. Andernfalls handelt es sich um einen Fehlerwert.

Hinweis Im Unterschied zur Funktion *RegCreateKey* erzeugt *RegOpenKey* den angegebenen Schlüssel nicht, wenn er noch nicht in der Datenbank existiert.

Beispiel Das folgende Beispiel verwendet *RegOpenKey*, um das Handle des untergeordneten Schlüssels StdFileEditing abzurufen, ruft dann die Funktion *RegQueryValue* auf, um den Namen eines Objekt-Handler abzurufen und verwendet anschließend die Funktion *RegDeleteKey*, um den Schlüssel zu löschen, wenn dieser den Wert »nwappobj.dll« hat:

```
char szBuff[80];
LONG cb;
HKEY hkStdFileEditing;

if (RegOpenKey(HKEY_CLASSES_ROOT,
        "NewAppDocument//protocol//StdFileEditing",
        &hkStdFileEditing) == ERROR_SUCCESS) {

    cb = sizeof(szBuff);
    if (RegQueryValue(hkStdFileEditing,
            "handler",
            szBuff,
```

```
                &cb) == ERROR_SUCCESS
                && lstrcmpi("nwappobj.dll", szBuff) == 0)
            RegDeleteKey(hkStdFileEditing, "handler");
        RegCloseKey(hkStdFileEditing);
}
```

Siehe auch *RegCreateKey*

RegQueryValue 3.1

Syntax

```
#include <shellapi.h>
LONG RegQueryValue(hkey, lpszSubKey, lpszValue, lpcb)

HKEY hkey;          /* Handle des zu untersuchenden
                       Schlüssels                    */
LPCSTR lpszSubKey;  /* Adresse des String für den
                       abzufragenden untergeordneten
                       Schlüssel                     */
LPSTR lpszValue;    /* Adresse des Puffers für den
                       zurückgegebenen String        */
LONG FAR* lpcb;     /* Adresse des Puffers für die Größe
                       des zurückgegebenen String    */
```

Beschreibung Die Funktion *RegQueryValue* ruft den Text-String ab, der dem angegebenen Schlüssel zugeordnet ist.

Parameter *hkey:* Bezeichnet einen aktuell geöffneten Schlüssel. Dabei kann es sich um *HKEY_CLASSES_ROOT* handeln. Dieser Wert darf nicht *NULL* sein.

lpszSubKey: Zeigt auf einen nullterminierten String, der den Namen des durch den Parameter *hkey* bezeichneten untergeordneten Schlüssels angibt, für den ein Text-String abgerufen werden soll. Wenn dieser Parameter *NULL* ist oder auf einen leeren String zeigt, ruft die Funktion den Wert des durch den Parameter *hkey* bezeichneten Schlüssels ab.

lpszValue: Zeigt auf einen Puffer, der bei der Rückkehr der Funktion den Text-String enthält.

lpcb: Zeigt auf eine Variable, die die Größe des durch den Parameter *lpszValue* bezeichneten Puffers in Byte enthält. Wenn die Funktion zurückkehrt, enthält diese Variable die Größe des in *lpszValue* kopierten String, einschließlich des nullterminierenden Zeichens.

Rückgabewert Ist die Funktion erfolgreich, ist der Rückgabewert *ERROR_SUCCESS*. Andernfalls handelt es sich um einen Fehlerwert.

Beispiel Das folgende Beispiel verwendet die Funktion *RegOpenKey*, um das Handle des untergeordneten Schlüssels StdFileEditing abzurufen. Dann wird die Funktion *RegQueryValue* aufgerufen,

um den Namen eines Objekt-Handler abzurufen und die Funktion *RegDeleteKey*, um den Schlüssel zu löschen, wenn er den Wert »nwappobj.dll« hat:

```
char szBuff[80];
LONG cb;
HKEY hkStdFileEditing;

if (RegOpenKey(HKEY_CLASSES_ROOT,
        "NewAppDocument\\protocol\\StdFileEditing",
        &hkStdFileEditing) == ERROR_SUCCESS) {

    cb = sizeof(szBuff);

if (RegQueryValue(hkStdFileEditing,
            "handler",
            szBuff,
            &cb) == ERROR_SUCCESS
            && lstrcmpi("nwappobj.dll", szBuff) == 0)
        RegDeleteKey(hkStdFileEditing, "handler");
    RegCloseKey(hkStdFileEditing);
}
```

Siehe auch *RegEnumKey*

RegSetValue 3.1

Syntax

```
#include <shellapi.h>
LONG RegSetValue(hkey, lpszSubKey, fdwType, lpszValue,
   cb)

HKEY hkey;          /* Handle des Schlüssels          */
LPCSTR lpszSubKey;  /* Adresse des String für den
                       untergeordneten Schlüssel      */
DWORD fdwType;      /* muß REG_SZ sein                */
LPCSTR lpszValue;   /* Adresse des String für den
                       Schlüssel                      */
DWORD cb;           /* wird ignoriert                 */
```

Beschreibung Die Funktion *RegSetValue* ordnet einem angegebenen Schlüssel einen Text-String zu.

Parameter *hkey:* Bezeichnet einen aktuell geöffneten Schlüssel. Dabei kann es sich um *HKEY_CLASSES_ROOT* handeln. Dieser Wert darf nicht *NULL* sein.

lpszSubKey: Zeigt auf einen nullterminierten String, der den untergeordneten Schlüssel des Parameters *hkey* bestimmt, dem der Text-String zugeordnet werden soll. Wenn dieser Parameter *NULL* ist oder auf einen leeren String zeigt, setzt die Funktion den Wert für den durch *hkey* bezeichneten Schlüssel.

fdwType: Bestimmt den String-Typ. Unter Windows 3.1 muß dieser Wert *REG_SZ* sein.

lpszValue: Zeigt auf einen nullterminierten String, der den für den angegebenen Schlüssel zu setzenden Text-String enthält.

cb: Bestimmt die Größe des durch den Parameter *lpszValue* bezeichneten Strings in Byte. Unter Windows 3.1 wird dieser Wert ignoriert.

Rückgabewert Ist die Funktion erfolgreich, ist der Rückgabewert *ERROR_SUCCESS*. Andernfalls handelt es sich um einen Fehlerwert.

Hinweis Wenn der durch den Parameter *lpszSubKey* angegebene Schlüssel nicht existiert, wird er von der Funktion *RegSetValue* erzeugt.

Beispiel Das folgende Beispiel verwendet die Funktion *RegSetValue*, um eine Dateinamenserweiterung und den zugeordneten Klassennamen zu registrieren:

```
RegSetValue(HKEY_CLASSES_ROOT, /* Hauptschlüssel */
    ".XXX",                    /* String mit der Datei-
                                  namenserweiterung
    REG_SZ,                    /* muß angegeben
                                  werden */
    "NewAppDocument",          /* Klassenname für die
                                  Namenserweiterung */
    14);                       /* Größe des Text-
                                  String */

RegSetValue(HKEY_CLASSES_ROOT, /* Hauptschlüssel */
    "NewAppDocument",          /* String mit dem
                                  Klassendefinitions-
                                  schlüssel */
    REG_SZ,                    /* muß angegeben
                                  werden */
    "New Application",         /* Beschreibung der
                                  Klasse */
    15);                       /* Größe des String */
```

Siehe auch *RegCreateKey, RegQueryValue*

ReplaceText

Syntax

```
HWND ReplaceText(lpfr)

LPFINDREPLACE lpfr;    /* Zeiger auf eine Struktur
                          mit Initialisierungsdaten */
```

Beschreibung Die Funktion *ReplaceText* erzeugt ein systemdefiniertes, nichtmodales Dialogfenster, mit dem der Anwender in einem Dokument einen Textausschnitt suchen und ersetzen kann. Die

Anwendung muß die eigentliche Suchen- und Ersetzungsoperation ausführen.

lpfr: Zeigt auf eine *FINDREPLACE*-Struktur, die Informationen zur Initialisierung des Dialogfensters enthält. Wenn der Anwender im Dialogfenster eine Auswahl trifft, füllt das System diese Struktur mit den entsprechenden Informationen und sendet eine Botschaft an die Anwendung. Diese Botschaft enthält einen Zeiger auf die *FINDREPLACE*-Datenstruktur.

Die Struktur *FINDREPLACE* hat folgende Form:

```
typedef struct tagFINDREPLACE  {   /* fr */
    DWORD lStructSize;
    HWND hwndOwner;
    HINSTANCE hInstance;
    DWORD Flags;
    LPSTR lpstrFindWhat;
    LPSTR lpstrReplaceWith;
    UINT wFindWhatLen;
    UINT wReplaceWithLen;
    LPARAM lCustData;
    UINT (CALLBACK)* lpfnHook) (HWND, UINT, WPARAM,
          LPARAM);
    LPCSTR lpTemplateName;
} FINDREPLACE;
```

Rückgabewert

Der Rückgabewert ist das Handle des Dialogfensters. Er ist *NULL,* falls ein Fehler aufgetreten ist. Eine Anwendung kann dieses Handle verwenden, um mit dem Dialogfenster zu kommunizieren oder um es zu schließen.

Fehler

Verwenden Sie die Funktion *CommDlgExtendedError,* um den Fehlerwert abzurufen. Möglich sind folgende Werte:

CDERR_FINDRESFAILURE
CDERR_INITIALISATION
CDERR_LOADRESFAILURE
CDERR_LOADSTRFAILURE
CDERR_LOCKRESFAILURE
CDERR_MEMALLOCFAILURE
CDERR_MEMLOCKFAILURE
CDERR_NOHINSTANCE
CDERR_NOHOOK
CDERR_NOTEMPLATE
CDERR_STRUCTSIZE
FRERR_BUFFERLENGTHZERO

Hinweis

Die Dialogfensterfunktion *ReplaceText* übergibt Anwenderanforderungen durch spezielle Botschaften an die Anwendung. Der Parameter *lParam* dieser Botschaften enthält einen Zeiger auf die *FINDREPLACE*-Struktur. Die Dialogfensterfunk-

tion sendet die Botschaften an das Fenster, das durch das Element *hwndOwner* der *FINDREPLACE*-Struktur bezeichnet wird. Eine Anwendung kann den Bezeichner für diese Botschaften registrieren, indem der String »*commdlg_FindReplace*« in einem Aufruf der Funktion *RegisterWindowMessage* angegeben wird.

Damit die Tabulatortaste richtig arbeitet, muß eine Anwendung außer der Funktion *ReplaceText* auch *IsDialogMessage* in ihrer Hauptbotschaftsschleife aufrufen. Die Funktion *IsDialogMessage* gibt einen Wert zurück, mit dessen Hilfe die Anwendung überprüfen kann, ob sich Botschaften auf das Dialogfenster *Ersetzen* beziehen.

Beispiel

Dieses Beispiel initialisiert eine *FINDREPLACE*-Struktur und ruft die Funktion *ReplaceText* auf, um das Dialogfenster *Ersetzen* anzuzeigen:

```
FINDREPLACE fr;
char szFindWhat[256] = "";     /* Such-String */
char szReplaceWith[256] = "";  /* Ersatz-String */

/* Alle Elemente der Struktur werden mit Null
   initialisiert */

memset(&fr, 0, sizeof(FINDREPLACE);

fr.lStructSize = sizeof(FINDREPLACE);
fr.hwndOwner = hwnd;
fr.lpstrFindWhat = szFindWhat;
fr.wFindWhatLen = sizeof(szFindWhat);
fr.lpstrReplaceWith = szReplaceWith);

hDlg = ReplaceText(&fr);
```

Zusätzlich zu der Initialisierung der Elemente der *FINDREPLACE*-Struktur muß eine Anwendung die *FINDMSGSTRING*-Botschaften registrieren und die Botschaften des Dialogfensters bearbeiten. In der Beschreibung der Funktion *FindText* finden Sie ein Beispiel für die Registrierung und Bearbeitung von Botschaften.

Querverweis

FindText, IsDialogMessage, RegisterWindowMessage

ResetDC

3.1

Syntax

```
#include <print.h>
HDC ResetDC(hdc, lpdm)

HDC hdc;                    /* Handle des Geräte-
                               kontexts              */
const DEVMODE FAR* lpdm;    /* Adresse der DEVMODE-
                               Struktur              */
```

Beschreibung Die Funktion *ResetDC* aktualisiert einen Gerätekontext mit den Informationen aus der angegebenen *DEVMODE*-Struktur.

Parameter *hdc:* Bezeichnet den zu aktualisierenden Gerätekontext.

lpdm: Zeigt auf eine *DEVMODE*-Struktur, die Informationen über den neuen Gerätekontext enthält. Die *DEVMODE*-Struktur hat folgende Form:

```
#include <print.h>

typedef struct tagDEVMODE {   /* dm */
    char dmDeviceName[CCHDEVICENAME];
    UINT dmSpecVersion;
    UINT dmDriverVersion;
    UINT dmSize;
    UINT dmDriverExtra;
    DWORD dmFields;
    int dmOrientation;
    int dmPaperSize;
    int dmPaperLength;
    int dmPaperWidth;
    int dmScale;
    int dmCopies;
    int dmDefaultSource;
    int dmPrintQuality;
    int dmColor;
    int dmDuplex;
    int dmYResolution;
    int dmTTOption;
} DEVMODE;
```

Rückgabewert Der Rückgabewert ist das Handle des ursprünglichen Gerätekontexts, wenn die Funktion erfolgreich ist. Andernfalls ist der Rückgabewert *NULL*.

Hinweis Anwendungen rufen die Funktion *ResetDC* normalerweise auf, wenn ein Fenster eine *WM_DEVMODECHANGE*-Botschaft erhält. *ResetDC* kann aber auch verwendet werden, um beim Drucken eines Dokuments die Druckrichtung zu ändern oder den Papiereinzugschacht zu wechseln.

Die Funktion *ResetDC* kann nicht zum Wechseln des Treibernamens, des Gerätenamens oder der Ausgabeschnittstelle verwendet werden. Ändert der Anwender die Schnittstellenverbindung oder den Gerätenamen, muß die Anwendung den ursprünglichen Gerätekontext löschen und einen neuen Gerätekontext mit den aktualisierten Informationen erstellen.

Vor einem Aufruf von *ResetDC* müssen Anwendungen die Auswahl aller Objekte (mit Ausnahme der vordefinierten Objekte) in den Gerätekontext aufheben.

Siehe auch *DeviceCapabilities, Escape, ExtDeviceMode*

ScaleViewportExtEx 3.1

Syntax

```
BOOL ScaleViewportExtEx(hdc, nXnum, nXdenom, nYnum,
   nYdenom, lpSize)

HDC hdc;          /* Handle des Gerätekontexts       */
int nXnum;        /* Betrag, mit dem die aktuelle
                     x-Abmessung multipliziert wird  */
int nXdenom;      /* Betrag, durch den die aktuelle
                     x-Abmessung dividiert wird      */
int nYnum;        /* Betrag, mit dem die aktuelle
                     y-Abmessung multipliziert wird  */
int nYdenom;      /* Betrag, durch den die aktuelle
                     y-Abmessung dividiert wird      */
SIZE FAR* lpSize; /* Adresse der SIZE-Struktur       */
```

Beschreibung

Die Funktion *ScaleViewportExtEx* ändert die Größe des Grafikfensters relativ zu den aktuellen Werten. Folgende Formeln werden verwendet:

```
xNewVE = (xOldVE * Xnum) / Xdenom
xNewVE = (yOldVE * Ynum) / Ydenom
```

Die neue Abmessung wird berechnet, indem die Werte der aktuellen Abmessung mit den angegebenen Nennern multipliziert und durch die angegebenen Teiler dividiert werden.

Parameter

hdc: Bezeichnet den Gerätekontext.

nXnum: Bestimmt den Betrag, mit dem die aktuelle x-Abmessung multipliziert wird.

nXdenum: Bestimmt den Betrag, durch den die aktuelle x-Abmessung dividiert wird.

nYnum: Bestimmt den Betrag, mit dem die aktuelle y-Abmessung multipliziert wird.

nYdenum: Bestimmt den Betrag, durch den die aktuelle y-Abmessung dividiert wird.

lpSize: Zeigt auf eine *SIZE*-Struktur. Die bisherigen Abmessungen des Grafikfensters werden in dieser Struktur in Geräteeinheiten abgelegt. Wenn *lpSize* den Wert *NULL* hat, wird nichts zurückgegeben.

Rückgabewert

Ist die Funktion erfolgreich, ist der Rückgabewert ungleich Null, andernfalls Null.

ScaleWindowExtEx 3.1

Syntax

```
int ScaleWindowExtEx(hdc, nXnum, nXdenom, nYnum,
   nYdenom, lpSize)
```

```
HDC hdc;          /* Handle des Gerätekontexts        */
int nXnum;        /* Betrag, mit dem die aktuelle
                     x-Abmessung multipliziert wird   */
int nXdenom;      /* Betrag, durch den die aktuelle
                     x-Abmessung dividiert wird       */
int nYnum;        /* Betrag, mit dem die aktuelle
                     y-Abmessung multipliziert wird   */
int nYdenom;      /* Betrag, durch den die aktuelle
                     y-Abmessung dividiert wird       */
SIZE FAR* lpSize; /* Adresse der SIZE-Struktur        */
```

Beschreibung Die Funktion *ScaleViewportExtEx* ändert die Größe eines Fensters relativ zu den aktuellen Werten. Folgende Formeln werden verwendet:

```
xNewVE = (xOldVE * Xnum) / Xdenom
xNewVE = (yOldVE * Ynum) / Ydenom
```

Die neue Abmessung wird berechnet, indem die Werte der aktuellen Abmessung mit den angegebenen Nennern multipliziert und durch die angegebenen Teiler dividiert werden.

Parameter *hdc:* Bezeichnet den Gerätekontext.

nXnum: Bestimmt den Betrag, mit dem die aktuelle x-Abmessung multipliziert wird.

nXdenum: Bestimmt den Betrag, durch den die aktuelle x-Abmessung dividiert wird.

nYnum: Bestimmt den Betrag, mit dem die aktuelle y-Abmessung multipliziert wird.

nYdenum: Bestimmt den Betrag, durch den die aktuelle y-Abmessung dividiert wird.

lpSize: Zeigt auf eine *SIZE*-Struktur. Die bisherigen Abmessungen des Fensters werden in dieser Struktur in Geräteeinheiten abgelegt. Wenn *lpSize* den Wert *NULL* hat, wird nichts zurückgegeben.

Rückgabewert Ist die Funktion erfolgreich, ist der Rückgabewert ungleich Null, andernfalls Null.

ScrollWindowEx 3.1

Syntax

```
int ScrollWindowEx(hwnd, dx, dy, lprcScroll, lprcClip,
   hrgnUpdate, lprcUpdate, fuScroll)

HWND hwnd;                /* Handle des Fensters, in
                             dem ein Bildlauf durch-
                             geführt werden soll    */
```

```
int dx;                     /* Betrag für den
                               horizontalen Bildlauf  */
int dy;                     /* Betrag für den vertikalen
                               Bildlauf               */
const RECT FAR* lprcScroll; /* Adresse der Struktur mit
                               dem Bildlaufrechteck   */
const RECT FAR* lprcClip;   /* Adresse der Struktur mit
                               dem Clipping-Rechteck  */
HRGN hrgnUpdate;            /* Handle der
                               Aktualisierungsregion  */
RECT FAR* lprcUpdate;       /* Adresse der Struktur für
                               die Aktualisierungs-
                               region                 */
UINT fuScroll;              /* Bildlauf-Flags         */
```

Beschreibung

Die Funktion *ScrollWindowEx* verschiebt den Inhalt des Client-Bereichs eines Fensters. Diese Funktion entspricht der Funktion *ScrollWindow*, besitzt aber einige zusätzliche Eigenschaften.

Parameter

hwnd: Bezeichnet das Fenster, in dem der Bildlauf durchgeführt werden soll.

dx: Bestimmt den Betrag für den horizontalen Bildlauf in Geräteeinheiten. Soll der Bildlauf nach links durchgeführt werden, muß ein negativer Wert angegeben werden.

dy: Bestimmt den Betrag für den vertikalen Bildlauf in Geräteeinheiten. Soll ein Bildlauf nach oben durchgeführt werden, muß ein negativer Wert angegeben werden.

lprcScroll: Zeigt auf eine *RECT*-Struktur, die den Teil des Client-Bereichs angibt, in dem der Bildlauf durchgeführt werden soll. Wenn dieser Parameter *NULL* ist, wird der gesamte Client-Bereich verschoben. Die *RECT*-Struktur hat folgende Form:

```
typedef stuct tagRECT {   /* rc */
    int left;
    int top;
    int right;
    int bottom;
} RECT;
```

lprcClip: Zeigt auf eine *RECT*-Struktur, die das Clipping-Rechteck angibt, in dem ein Bildlauf durchgeführt werden soll. Diese Struktur hat Vorrang vor der Struktur, auf die der Parameter *lprcScroll* zeigt. Ein Bildlauf wird nur für Bits in diesem Rechteck durchgeführt. Die Bits außerhalb des Rechtecks werden nicht beeinflußt, auch wenn sie sich im Rechteck *lprcScroll* befinden. Wenn dieser Parameter *NULL* ist, wird ein Bildlauf für den gesamten Client-Bereich durchgeführt.

hrgnUpdate: Bezeichnet die Region, die durch den Bildlauf ungültig wird. Dieser Parameter kann *NULL* sein.

lprcUpdate: Zeigt auf eine *RECT*-Struktur, die die Koordinaten des umgebenden Rechtecks des Bereichs aufnimmt, der durch den Bildlauf ungültig geworden ist. Dieser Parameter kann *NULL* sein.

fuScroll: Bestimmt Flags, die den Bildlauf steuern. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
SW_ERASE	Wird dieser Wert zusammen mit *SW_INVALIDATE* angegeben, wird die ungültig gewordene Region durch die Übergabe einer *WM_ERASEBKGND*-Botschaft an das Fenster gelöscht.
SW_INVALIDATE	Die durch den Parameter *hrgnUpdate* bezeichnete Region wird nach dem Bildlauf ungültig.
SW_SCROLLCHILDREN	In allen untergeordneten Fenstern, die sich mit dem durch den Parameter *lprcScroll* bezeichneten Rechteck überschneiden, wird ein Bildlauf um die in den Parametern *dx* und *dy* angegebene Anzahl Pixel durchgeführt. Windows übergibt eine *WM_MOVE*-Botschaft an alle untergeordneten Fenster, die sich mit dem durch *lprcScroll* angegebenen Rechteck überschneiden, auch wenn sie sich nicht verändern. Das Caret wird neu positioniert, wenn ein Bildlauf in einem untergeordneten Fenster stattfindet und sich das Mauszeigerrechteck mit dem Bildlaufrechteck überschneidet.

Rückgabewert

Der Rückgabewert ist *SIMPLEREGION* (die ungültig gewordene Region ist rechteckig), *COMPLEXREGION* (die ungültig gewordene Region ist nicht rechteckig; überlappende Rechtecke) oder *NULLREGION* (keine ungültig gewordene Region), wenn die Funktion erfolgreich ist. Andernfalls wird der Wert *ERROR* zurückgegeben.

Hinweis

Sind die Flags *SW_INVALIDATE* und *SW_ERASE* nicht angegeben, wird der Bereich, dessen Inhalt durch *ScrollWindowEx* hinausgeschoben wurde, nicht ungültig. Ist eines dieser Flags gesetzt, macht *ScrollWindowEx* diesen Bereich ungültig. Der Bereich wird erst aktualisiert, wenn die Anwendung die Funktion *UpdateWindow* oder die Funktion *RedrawWindow* mit *RDW_UPDATENOW* oder *RDW_ERASENOW* aufruft oder eine *WM_PAINT*-Botschaft aus der Warteschlange der Anwendung erhält.

Hat das Fenster den Stil *WS_CLIPCHILDREN*, repräsentieren die in *hrgnUpdate* und *lprcUpdate* zurückgegebenen Bereiche den Gesamtbereich des Fensters, der aktualisiert werden muß. Darin eingeschlossen sind die Bereiche untergeordneter Fenster, die aktualisiert werden müssen.

Ist das Flag *SW_SCROLLCHILDREN* angegeben, kann Windows den Bildschirm nicht korrekt aktualisieren, wenn nur in einem Teil eines untergeordneten Fensters ein Bildlauf durchgeführt wird. Der außerhalb des Quellrechtecks liegende Teil des entsprechenden untergeordneten Fensters wird nicht gelöscht. Es wird außerdem nicht korrekt an seiner neuen Position neu gezeichnet. Verwenden Sie die Funktion *DeferWindowPos*, um die untergeordneten Fenster in das *lprcScroll*-Rechteck zu verlagern.

Alle Eingabe- und Ausgabekoordinaten für *lprcScroll*, *lprcClip*, *lprcUpdate* und *hrgnUpdate* sind Client-Koordinaten, unabhängig davon, ob das Fenster dem Klassenstil *CS_OWNDC* oder *CS_CLASSDC* hat. Verwenden Sie die Funktionen *LPtoDP* und *DPtoLP*, um in und aus virtuellen Koordinaten zu konvertieren.

Siehe auch *RedrawWindow, ScrollDC, ScrollWindow, UpdateWindow*

SendDriverMessage 3.1

Syntax

```
LRESULT SendDriverMessage(hdrvr, msg, lParam1, lParam2)

HDRVR hdrvr;     /* Handle eines installierbaren
                    Treibers                        */
UINT msg;        /* Botschaft                       */
LPARAM lParam1;  /* Erster Botschaftsparameter      */
LPARAM lParam2;  /* Zweiter Botschaftsparameter     */
```

Beschreibung Die Funktion *SendDriverMessage* übergibt die angegebene Botschaft an den installierbaren Treiber.

Parameter *hdrvr:* Bezeichnet den installierbaren Treiber.

msg: Bestimmt die Botschaft, die der Treiber bearbeiten muß. Folgende Botschaften sollten niemals von Anwendungen direkt an den Treiber übergeben werden. Sie werden nur vom System übergeben:

DRV_CLOSE
DRV_DISABLE
DRV_ENABLE
DRV_EXITAPPLICATION
DRV_EXITSESSION
DRV_FREE
DRV_LOAD
DRV_OPEN

lParam1: Bestimmt 32 Bits zusätzlicher, botschaftsabhängiger Informationen.

lParam2: Bestimmt 32 Bits zusätzlicher, botschaftsabhängiger Informationen.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Siehe auch *DefDriverProc*

SetAbortProc 3.1

Syntax

```
int SetAbortProc(hdc, abrtprc)

HDC hdc;            /* Handle des Gerätekontexts     */
ABORTPROC abrtprc;  /* Adresse der Instanz der
                       Abbruchfunktion               */
```

Beschreibung Die Funktion *SetAbortProc* installiert die anwendungsdefinierte Funktion, die das Abbrechen von Druckaufträgen während ihrer Aufzeichnung ermöglicht. Diese Funktion ersetzt den Drucker-Escape-Code *SETABORTPROC*.

Parameter *hdc:* Bezeichnet den Gerätekontext für den Druckauftrag.

abrtprc: Bestimmt die Adresse der Prozedurinstanz der Callback-Funktion. Die Adresse muß mit der Funktion *MakeProcInstance* erzeugt worden sein. Informationen über die Callback-Funktion finden Sie in der Beschreibung von *AbortProc*.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert größer als Null, andernfalls kleiner als Null.

Siehe auch *AbortDoc, AbortProc, Escape*

SetBitmapDimensionEx 3.1

Syntax

```
BOOL SetBitmapDimensionEx(hbm, nX, nY, lpSize)

HBITMAP hbm;       /* Handle des Bitmap              */
int nX;            /* Breite des Bitmap              */
int nY;            /* Höhe des Bitmap                */
SIZE FAR* lpSize;  /* Adresse der Struktur für die
                      vorherigen Abmessungen          */
```

Beschreibung Die Funktion *SetBitmapDimensionEx* weist dem angegebenen Bitmap die gewünschte Größe in Schritten von 0.1 Millimeter zu. Die grafische Geräteschnittstelle (GDI) verwendet diese Werte nicht. Sie gibt sie nur bei einem Aufruf der Funktion *GetBitmapDimensionEx* an die Anwendung weiter.

Parameter *hbm:* Bezeichnet das Bitmap.

nX: Bestimmt die Breite des Bitmap in Einheiten von 0.1 Millimeter.

nY: Bestimmt die Höhe des Bitmap in Einheiten von 0.1 Millimeter.

lpSize: Zeigt auf eine *SIZE*-Struktur. Die vorherigen Abmessungen des Bitmap werden in dieser Struktur abgelegt. Wenn *lpSize* den Wert *NULL* hat, wird nichts zurückgegeben. Die *SIZE*-Struktur hat folgende Form:

```
typedef struct tagSIZE {
    int cx;
    int cy;
} SIZE;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

SetBoundsRect 3.1

Syntax

```
UINT SetBoundsRect (hdc, lprcBounds, flags)

HDC hdc;                    /* Handle des
                               Gerätekontexts        */
const RECT FAR* lprcBounds; /* Adresse der Struktur für
                               das Rechteck          */
UINT flags;                 /* Enthält die zurückzu-
                               gebenden Informationen */
```

Beschreibung Die Funktion *SetBoundsRect* steuert die Akkumulation der Informationen über das umgebenden Rechteck für den angegebenen Gerätekontext.

Parameter *hdc:* Bezeichnet den Gerätekontext, für den die Informationen über das umgebende Rechteck akkumuliert werden sollen.

lprcBounds: Zeigt auf eine *RECT*-Struktur, die zum Setzen des umgebenden Rechtecks benutzt wird. Die Abmessungen des Rechtecks werden in virtuellen Koordinaten angegeben. *lprcBounds* kann *NULL* sein. Die *RECT*-Struktur hat folgende Form:

```
typedef struct tagRECT {   /* rc */
    int left;
    int top;
    int right;
    int bottom;
} RECT;
```

flags: Bestimmt, wie das neue Rechteck mit dem bereits akkumulierten Rechteck kombiniert wird. Dieser Parameter kann eine Kombination der folgenden Werte sein:

Wert	Bedeutung
DCB_ACCUMULATE	Das im Parameter *lprcBounds* angegebene Rechteck wird unter Verwendung der Vereinigungsoperation für Rechtecke zum umgebenden Rechteck hinzugefügt.
DCB_DISABLE	Schaltet die Akkumulation für das umgebende Rechteck ab.
DCB_ENABLE	Schaltet die Akkumulation für das umgebende Rechteck ein. In der Voreinstellung ist die Akkumulation des umgebenden Rechtecks deaktiviert.
DCB_RESET	Setzt ein leeres umgebendes Rechteck.
DCB_SET	Setzt das umgebende Rechteck auf die Koordinaten, die im Parameter *lprcBounds* angegeben sind.

Rückgabewert Der Rückgabewert ist der aktuelle Status des umgebenden Rechtecks, wenn die Funktion erfolgreich ist. Wie der Parameter *flags* kann auch der Rückgabewert eine Kombination der *DCB_-Werte* sein.

Hinweis Windows kann für alle Zeichenoperationen ein gemeinsames umgebendes Rechteck verwalten. Es kann durch die Anwendung abgefragt und zurückgesetzt werden. Dieses umgebende Rechteck ist hilfreich, wenn der Bitmap-Cache ungültig werden soll.

Anwendungen sollten die Flags *DCB_ACCUMULATE* und *DCB_RESET* im Rückgabewert überprüfen, um sicherzustellen, ob das Rechteck leer ist. Wenn das Flag *DCB_RESET* gesetzt und das Flag *DCB_ACCUMULATE* nicht gesetzt ist, ist das umgebende Rechteck leer.

Siehe auch *GetBoundsRect*

SetMetaFileBitsBetter 3.1

Syntax
```
HGLOBAL SetMetaFileBitsBetter(hmf)
HMETAFILE hmf;   /* Handle des Metafile                */
```

Beschreibung Die Funktion *SetMetaFileBitsBetter* erzeugt ein Speicher-Metafile aus den Daten im angegebenen globalen Speicherobjekt.

Parameter *hmf:* Bezeichnet das globale Speicherobjekt, das die Metafile-Daten enthält. Das Objekt muß durch einen vorhergehenden Aufruf der Funktion *GetMetaFileBits* erzeugt worden sein.

Rückgabewert Der Rückgabewert ist das Handle eines Speicher-Metafile, wenn die Funktion erfolgreich ist, andernfalls ist er *NULL*.

Hinweis Das von der Funktion *SetMetaFileBitsBetter* zurückgegebene Handle des globalen Speicherobjekts ist der GDI zugeordnet, nicht der Anwendung. So können Anwendungen, die Metafiles für OLE unterstützen, Metafiles verwenden, die auch nach der Beendigung einer Anwendung bestehenbleiben. Eine OLE-Anwendung sollte immer die Funktion *SetMetaFileBitsBetter* anstelle von *SetMetaFileBits* verwenden.

Nachdem die Funktion *SetMetaFileBitsBetter* zurückgekehrt ist, sollte das zurückgegebene Metafile-Handle verwendet werden, um das Metafile zu referenzieren.

Das durch den Parameter *hmf* bezeichnete Handle sollte nicht verwendet werden.

Siehe auch *GetMetaFileBits*, *SetMetaFileBits*

SetSelectorBase 3.1

Syntax

```
UINT SetSelectorBase(selector, dwBase)

UINT selector;   /* neuer Selektor                */
DWORD dwBase;    /* neue Basis                    */
```

Beschreibung Die Funktion *SetSelectorBase* setzt Basis und Grenze eines Selektors.

Parameter *selector:* Bestimmt den neuen Selektorwert.

dwBase: Bestimmt den neuen Basiswert.

Rückgabewert Der Rückgabewert ist der neue Selektorwert, wenn die Funktion erfolgreich ist.

Siehe auch *GetSelectorBase*, *GetSelectorLimit*, *SetSelectorLimit*

SetSelectorLimit 3.1

Syntax

```
UINT SetSelectorLimit(selector, dwLimit)

UINT selector;   /* neuer Selektor                */
DWORD dwLimit;   /* aktuelle Grenze               */
```

Beschreibung Die Funktion *SetSelectorLimit* setzt den Grenzwert eines Selektors.

Parameter *selector:* Bestimmt den neuen Selektorwert.

dwBase: Bestimmt den Grenzwert für den Selektor.

Rückgabewert Der Rückgabewert ist immer Null.

Siehe auch *GetSelectorBase*, *GetSelectorLimit*, *SetSelectorBase*

SetViewportExtEx 3.1

Syntax

```
BOOL SetViewportExtEx(hdc, nX, nY, lpSize)

HDC hdc;          /* Handle des Gerätekontexts        */
int nX;           /* x-Abmessung des Grafikfensters   */
int nY;           /* y-Abmessung des Grafikfensters   */
SIZE FAR* lpSize; /* Adresse der Struktur mit den
                     vorherigen Abmessungen           */
```

Beschreibung *SetViewportExtEx* setzt die x- und y-Abmessungen des Grafikfensters des angegebenen Gerätekontexts. Das Grafikfenster definiert zusammen mit dem Fenster, wie Punkte aus virtuellen Koordinaten auf Gerätekoordinaten abgebildet werden.

Parameter *hdc:* Bezeichnet den Gerätekontext.

nX: Bestimmt die x-Abmessung des Grafikfensters in Geräteeinheiten.

nY: Bestimmt die y-Abmessung des Grafikfensters in Geräteeinheiten.

lpSize: Zeigt auf eine *SIZE*-Struktur. Die vorherigen Abmessungen des Grafikfensters werden in Geräteeinheiten in dieser Struktur abgelegt. Wenn *lpSize NULL* ist, wird nichts zurückgegeben. Die *SIZE*-Struktur hat folgende Form:

```
typedef struct tagSIZE {
    int cx;
    int cy;
} SIZE;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Ist einer der folgenden Abbildungsmodi gesetzt, werden Aufrufe der Funktionen *SetWindowExtEx* und *SetViewportExtEx* ignoriert:

MM_HIENGLISH
MM_HIMETRIC
MM_LOENGLISH
MM_LOMETRIC
MM_TEXT
MM_TWIPS

Wenn der Modus *MM_ISOTROPIC* gesetzt ist, müssen Anwendungen die Funktion *SetWindowExtEx* vor der Funktion *SetViewportExtEx* aufrufen.

Siehe auch *SetWindowExtEx*

SetViewportOrgEx 3.1

Syntax

```
BOOL SetViewportOrgEx(hdc, nX, nY, lpPoint)

HDC hdc;            /* Handle des Gerätekontexts      */
int nX;             /* x-Koordinate des neuen
                       Ursprungs                      */
int nY;             /* y-Koordinate des neuen
                       Ursprungs                      */
POINT FAR* lpPoint; /* Adresse der Struktur für den
                       vorherigen Ursprung            */
```

Beschreibung Die Funktion *SetViewportOrgEx* setzt den Ursprung des Grafikfensters im angegebenen Gerätekontext. Das Grafikfenster definiert zusammen mit dem Fenster, wie virtuelle Koordinaten auf Gerätekoordinaten abgebildet werden.

Parameter *hdc:* Bezeichnet den Gerätekontext.

nX: Bestimmt die x-Koordinate des Ursprungs des Grafikfensters in Geräteeinheiten.

nY: Bestimmt die y-Koordinate des Ursprungs des Grafikfensters in Geräteeinheiten.

lpPoint: Zeigt auf eine *POINT*-Struktur. Der vorhergehende Ursprung des Grafikfensters wird in dieser Struktur in Gerätekoordinaten abgelegt. Wenn *lpPoint NULL* ist, wird nichts zurückgegeben.

Die *POINT*-Struktur hat folgende Form:

```
typedef struct tagPOINT {   /* pt */
    int x;
    int y;
} POINT;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Siehe auch *SetWindowOrgEx*

SetWinDebugInfo 3.1

Syntax

```
BOOL SetWinDebugInfo(lpwdi)

const WINDEBUGINFO FAR* lpwdi;   /* Adresse der
                                    WINDEBUGINFO-
                                    Struktur        */
```

Beschreibung Die Funktion *SetDebugWindowInfo* initialisiert die aktuellen Debug-Informationen des Systems für die Debugger-Version von Windows 3.1.

Parameter *lpwdi:* Zeigt auf eine *WINDEBUGINFO*-Struktur, die den Typ der zu initialisierenden Debug-Informationen bestimmt. Die *WINDEBUGINFO*-Struktur hat folgende Form:

```
typedef struct tagWINDEBUGINFO {
    UINT flags;
    DWORD dwOptions;
    DWORD dwFilter;
    char achAllocModule[8];
    DWORD dwAllocBreak;
    DWORD dwAllocCount;
} WINDEBUGINFO;
```

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich ist. Er ist Null, wenn der im Parameter *lpwdi* angegebene Zeiger oder das Element *flags* der *WINDEBUGINFO*-Struktur ungültig ist. Der Rückgabewert ist auch dann Null, wenn die Funktion nicht in der Debugger-Version von Windows 3.1 aufgerufen wird.

Hinweis Das Element *flags* der *WINDEBUGINFO*-Struktur bestimmt, welche Debug-Informationen initialisiert werden sollen. Anwendungen müssen nur diejenigen Elemente der *WINDEBUGINFO* initialisieren, die den im Element *flags* gesetzten Flags entsprechen.

Änderungen der Debug-Informationen durch Aufruf der Funktion *SetWinDebugInfo* werden erst wirksam, wenn Sie das System verlassen oder den Computer neu starten.

Siehe auch *GetWinDebugInfo*

SetWindowExtEx 3.1

Syntax

```
BOOL SetWindowExtEx(hdc, nX, nY, lpSize)

HDC hdc;           /* Handle des Gerätekontexts       */
int nX;            /* x-Abmessung des Fensters        */
int nY;            /* y-Abmessung des Fensters        */
SIZE FAR* lpSize;  /* Adresse der Struktur mit den
                      vorherigen Abmessungen des
                      Fensters                        */
```

Beschreibung Die Funktion *SetWindowExtEx* setzt die x- und y-Abmessungen des Fensters, das dem angegebenen Gerätekontext zugeordnet ist. Das Fenster definiert zusammen mit dem Grafikfenster, wie Punkte aus virtuellen Koordinaten auf Gerätekoordinaten abgebildet werden.

Parameter *hdc:* Bezeichnet den Gerätekontext.

nX: Bestimmt die x-Abmessung des Fensters in virtuellen Einheiten.

nY: Bestimmt die y-Abmessung des Fensters in virtuellen Einheiten.

lpSize: Zeigt auf eine *SIZE*-Struktur. Die vorhergehenden Abmessungen des Fensters werden in dieser Struktur in virtuellen Einheiten abgelegt. Wenn *lpSize NULL* ist, wird nichts zurückgegeben.

Die *SIZE*-Struktur hat folgende Form:

```
typedef struct tagSIZE {
    int cx;
    int cy;
} SIZE;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Ist einer der folgenden Abbildungsmodi gesetzt, werden Aufrufe der Funktionen *SetWindowExtEx* und *SetViewportExtEx* ignoriert:

MM_HIENGLISH
MM_HIMETRIC
MM_LOENGLISH
MM_LOMETRIC
MM_TEXT
MM_TWIPS

Ist der Modus *MM_ISOTROPIC* gesetzt, müssen Anwendungen die Funktion *SetWindowExtEx* vor *SetViewportExtEx* aufrufen.

Siehe auch *SetViewportExtEx*

SetWindowOrgEx 3.1

Syntax

```
BOOL SetWindowOrgEx(hdc, nX, nY, lpPoint)

HDC hdc;             /* Handle des Gerätekontexts     */
int nX;              /* x-Koordinate des Fensters     */
int nY;              /* y-Koordinate des Fensters     */
POINT FAR* lpPoint;  /* Adresse der Struktur mit dem
                        vorherigen Ursprung           */
```

Beschreibung Die Funktion *SetWindowOrgEx* setzt den Ursprung des Fensters des angegebenen Gerätekontexts. Das Fenster definiert zusammen mit dem Grafikfensters wie Punkte in virtuellen Koordinaten auf Gerätekoordinaten abgebildet werden.

Parameter *hdc:* Bezeichnet den Gerätekontext.

nX: Bestimmt die virtuelle x-Koordinate des neuen Ursprungs des Fensters.

nY: Bestimmt die virtuelle y-Koordinate des neuen Ursprungs des Fensters.

lpPoint: Zeigt auf eine *POINT*-Struktur. Der vorhergehende Ursprung des Fensters wird in dieser Struktur abgelegt. Wenn *lpPoint NULL* ist, wird nichts zurückgegeben. Die *POINT*-Struktur hat folgende Form:

```
typedef struct tagPOINT {   /* pt */
    int x;
    int y;
} POINT;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Siehe auch *SetViewportOrgEx*

SetWindowPlacemet 3.1

Syntax

```
BOOL SetWindowPlacement(hwnd, lpwndpl)

HWND hwnd;                  /* Handle des Fensters     */
const WINDOWPLACEMENT FAR* lpwndpl;
                            /* Adresse der Struktur
                               mit den Positionsdaten  */
```

Beschreibung Die Funktion *SetWindowPlacement* setzt den Anzeigestatus und die Positionen für die normale (wiederhergestellte), die Vollbild- und die Symboldarstellung eines Fensters.

Parameter *hwnd:* Bezeichnet das Fenster.

lpwndpl: Zeigt auf eine *WINDOWPLACEMENT*-Struktur, die den neuen Anzeigestatus und die Positionen des Fensters bestimmt.

Die *WINDOWPLACEMENT*-Struktur hat folgende Form:

```
typedef struct tagWINDOWPLACEMENT {   /* wndpl */
    UINT length;
    UINT flags;
    UINT showCmd;
    POINT ptMinPosition;
    POINT ptMaxPosition;
    RECT rcNormalPosition;
} WINDOWPLACEMENT;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Siehe auch *GetWindowPlacement*

SetWindowsHookEx 3.1

Syntax

```
HHOOK SetWindowsHookEx(idHook, hkprc, hinst, htask)

int idHook;       /* Typ des zu installierenden
                     Filters                        */
HOOKPROC hkprc;   /* Adresse der Prozedurinstanz der
                     Filterfunktion                 */
HINSTANCE hinst;  /* Handle der Anwendungsinstanz   */
HTASK htask;      /* Task, für die der Filter
                     installiert werden soll        */
```

Beschreibung *SetWindowsHookEx* installiert eine anwendungsdefinierte Filterfunktion in einer Kette von Filterfunktionen. Diese Funktion ist eine erweiterte Version der Funktion *SetWindowsHook*.

Parameter *idHook:* Bestimmt den Typ des Filters, der installiert werden soll. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
WH_CALLWNDPROC	Installiert einen Fensterfunktionsfilter. Weitere Informationen finden Sie in der Beschreibung der Callback-Funktion *CallWndProc*.
WH_CBT	Installiert einen CBT-Filter (Computer Based Training). Weitere Informationen finden Sie in der Beschreibung der Callback-Funktion *CBTProc*.
WH_DEBUG	Installiert einen Debug-Filter. Weitere Informationen finden Sie in der Beschreibung der Callback-Funktion *DebugProc*.
WH_GETMESSAGE	Installiert einen Botschaftsfilter. Weitere Informationen finden Sie in der Beschreibung der Callback-Funktion *GetMsgProc*.
WH_HARDWARE	Installiert einen Botschaftsfilter für nicht standardisierte Hardware. Weitere Informationen finden Sie in der Beschreibung der Callback-Funktion *HardwareProc*.
WH_JOURNALPLAYBACK	Installiert einen Wiedergabeprotokollfilter. Weitere Informationen finden Sie in der Beschreibung der Callback-Funktion *JournalPlaybackProc*.
WH_JOURNALRECORD	Installiert einen Aufzeichnungsprotokollfilter. Weitere Informationen finden Sie in der Beschreibung der Callback-Funktion *JournalRecordProc*.

Wert	Bedeutung
WH_KEYBOARD	Installiert einen Tastaturfilter. Weitere Informationen finden Sie in der Beschreibung der Callback-Funktion *KeyboardProc*.
WH_MOUSE	Installiert einen Mausbotschaftsfilter. Weitere Informationen finden Sie in der Beschreibung der Callback-Funktion *MouseProc*.
WH_MSGFILTER	Installiert einen Botschaftsfilter. Weitere Informationen finden Sie in der Beschreibung der Callback-Funktion *MessageProc*.
WH_SYSMSGFILTER	Installiert einen systemweiten Botschaftsfilter. Weitere Informationen finden Sie in der Beschreibung der Callback-Funktion *SysMsgProc*.

hkprc: Bestimmt die Adresse der Prozedurinstanz der zu installierenden anwendungsdefinierten Filterfunktion.

hinst: Bezeichnet die Instanz des Moduls, das die Filterfunktion enthält.

htask: Bezeichnet die Task, für die der Filter installiert werden soll. Wenn dieser Parameter *NULL* ist, bezieht sich die installierte Filterfunktion auf das gesamte System und kann in jedem Kontext aus jeder Task aufgerufen werden.

Rückgabewert

Der Rückgabewert ist ein Handle des installierten Filters, wenn die Funktion erfolgreich ist. Die Anwendung oder die Bibliothek muß dieses Handle benutzen, um den Filter beim Aufruf der Funktion *CallNextHookEx* oder *UnhookWindowsHookEx* zu bezeichnen. Tritt ein Fehler auf, dann ist der Rückgabewert *NULL*.

Hinweis

Anwendungen oder Bibliotheken können die Funktion *GetCurrentTask* oder *GetWindowTask* benutzen, um Task-Handles für Filter in einer bestimmten Task abzurufen.

Filterfunktionen, die mit *SetWindowsHookEx* benutzt werden, müssen so deklariert werden:

```
DWORD HookProc(code, wParam, lParam)
int code;
WORD wParam;
LONG lParam;
{
    if (...)
        return CallNextHookEx(hhook, code, wParam,
               lParam);
}
```

Die Weitergabe an die nächste Filterfunktion – also der Aufruf der Funktion *CallNextHookProc* – ist optional. Anwendungen oder Bibliotheken können die nächste Filterfunktion vor oder nach der Bearbeitung in ihrer eigenen Filterfunktion aufrufen.

Vor Beendigung einer Anwendung muß diese die Funktion *UnhookWindowsHookEx* aufrufen, um die dem Filter zugeordneten Systemressourcen freizugeben.

Einige Filter können nur für das gesamte System gesetzt werden, andere für eine bestimmte Task. Einzelheiten entnehmen Sie der folgenden Liste:

Filter	Gültigkeitsbereich
WH_CALLWNDPROC	Task oder System
WH_CBT	Task oder System
WH_DEBUG	Task oder System
WH_GETMESSAGE	Task oder System
WH_HARDWARE	Task oder System
WH_JOURNALRECORD	System
WH_JOURNALPLAYBACK	System
WH_KEYBOARD	Task oder System
WH_MOUSE	Task oder System
WH_MSGFILTER	Task oder System
WH_SYSMSGFILTER	System

Für einen gegebenen Filtertyp werden zuerst die Filter der Task aufgerufen, anschließend die Filter des Systems.

Der Filter *WH_CALLWNDPROC* beeinflußt die Geschwindigkeit des Systems. Er wurde für Debug-Zwecke bereitgestellt.

Die Systemfilter sind gemeinsam genutzte Ressourcen. Das Installieren dieser Filter beeinflußt also alle Anwendungen. Alle Systemfilterfunktionen müssen sich in Bibliotheken befinden. Systemfilter sollten auf spezielle Anwendungen oder auf die Verwendung als Hilfe bei der Entwicklung von Programmen beschränkt werden. Benötigt eine Bibliothek einen bestimmten Filter nicht mehr, sollte sie die Filterfunktion löschen.

Task-Filter haben mehrere Vorteile gegenüber Systemfiltern. Sie bedingen keinen systemweiten Overhead in Anwendungen, die von ihrem Aufruf nicht betroffen sind. Die Implementation einer Filterfunktion für eine bestimmte Task muß nicht in einer separaten dynamischen Linkbibliothek erfolgen. Außerdem werden task-spezifische Filterfunktionen auch dann noch arbeiten, wenn zukünftige Versionen von Windows das Installieren von Systemfiltern aus Sicherheitsgründen nicht zulassen.

Um eine Filterfunktion zu installieren, muß der Funktion *SetWindowsHookEx* die Adresse der Prozedurinstanz dieser

Funktion übergeben werden. Die Funktion selbst muß in der Moduldefinitionsdatei der Bibliothek exportiert werden. Bibliotheken können die Prozeduradresse direkt übergeben. Tasks müssen die Funktion *MakeProcInstance* benutzen, um die Adresse der Prozedurinstanz zu ermitteln. Dynamische Linkbibliotheken müssen die Funktion *GetProcAddress* benutzen, um die Adresse der Prozedurinstanz zu erhalten.

Für einen gegebenen Filtertyp werden zuerst die Filter der Task aufgerufen, anschließend die Systemfilter.

Die Filter *WH_SYSMSGFILTER* werden vor den Filtern *WH_MSGFILTER* aufgerufen. Gibt eine der *WH_SYSMSGFILTER*-Filterfunktionen *TRUE* zurück, werden die *WH_MSGFILTER*-Filter nicht aufgerufen.

Siehe auch *CallNextHookEx*, *GetProcAddress*, *MakeProcInstance*, *MessageBox*, *PeekMessage*, *PostMessage*, *SendMessage*, *UnhookWindowsHookEx*

ShellExecute 3.1

Syntax

```
#include <shellapi.h>
HINSTANCE ShellExecute(hwnd, lpszOp, lpszFile,
   lpszParams, lpszDir, fsShowCmd)

HWND hwnd;          /* Handle des übergeordneten
                       Fensters                        */
LPCSTR lpszOp;      /* Adresse des String für die
                       durchzuführende Operation       */
LPCSTR lpszFile;    /* Adresse des String für den
                       Dateinamen                      */
LPCSTR lpszParams; /* Adresse des String für die
                       Parameter der ausführbaren Datei */
LPCSTR lpszDir;     /* Adresse des String für das
                       Verzeichnis                     */
int fsShowCmd;      /* Datei beim ¹ffnen anzeigen?     */
```

Beschreibung Die Funktion *ShellExecute* öffnet oder druckt die angegebene Datei.

Parameter *hwnd:* Bezeichnet das übergeordnete Fenster. Dieses Fenster empfängt alle Meldungsfenster, die von einer Anwendung erzeugt werden, um z. B. Fehler anzuzeigen.

lpszOp: Zeigt auf einen nullterminierten String, der die auszuführende Operation bestimmt. Dieser String kann "open" oder "print" lauten. Wenn der Parameter *NULL* ist, ist "open" der voreingestellte Wert.

lpszFile: Zeigt auf einen nullterminierten String, der die zu öffnende Datei bestimmt.

lpszParams: Zeigt auf einen nullterminierten String, der die Parameter bestimmt. Diese Parameter werden an die Anwendung übergeben, wenn der Parameter *lpszFile* eine ausführbare Datei bezeichnet. Zeigt *lpszFile* auf einen String, der ein Dokument bestimmt, hat dieser Parameter den Wert *NULL*.

lpszDir: Zeigt auf einen nullterminierten String, der das Arbeitsverzeichnis bestimmt.

fsShowCmd: Bestimmt, ob das Anwendungsfenster dargestellt werden soll, wenn die Anwendung geöffnet wird. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
SW_HIDE	Verbirgt das Fenster und aktiviert ein anderes Fenster.
SW_MINIMIZE	Verkleinert das angegebene Fenster zum Symbol und aktiviert das Fenster oberster Ebene in der Fensterliste des Systems.
SW_RESTORE	Aktiviert ein Fenster und stellt es dar. Wenn das Fenster als Vollbild oder als Symbol dargestellt ist, stellt Windows es in seiner ursprünglichen Größe und an der ursprünglichen Position wieder her (entspricht *SW_SHOWNORMAL*).
SW_SHOW	Aktiviert ein Fenster und stellt es in seiner aktuellen Größe und an der aktuellen Position dar.
SW_SHOW-MAXIMIZED	Aktiviert ein Fenster und stellt es als Vollbild dar.
SW_SHOW-MINIMIZE	Aktiviert ein Fenster und stellt es als Symbol dar.
SW_SHOW-MINNOACTIVE	Stellt ein Fenster als Symbol dar. Das aktuell aktive Fenster bleibt aktiv.
SW_SHOWNA	Stellt ein Fenster mit seinem aktuellen Status dar. Das aktuell aktive Fenster bleibt aktiv.
SW_SHOWNO-ACTIVATE	Stellt ein Fenster in der zuletzt verwendeten Größe und an der letzten Position dar. Das aktuell aktive Fenster bleibt aktiv.
SW_SHOW-NORMAL	Aktiviert ein Fenster und stellt es dar. Wenn das Fenster als Symbol oder als Vollbild dargestellt wird, stellt Windows es in seiner ursprünglichen Größe und an seiner ursprünglichen Position dar (entspricht *SW_RESTORE*).

Rückgabewert Der Rückgabewert ist das Instanz-Handle der Anwendung, die geöffnet oder gedruckt wurde, wenn die Funktion erfolgreich ist. Dieses Handle kann auch das Handle einer DDE-Server-Anwendung sein. Ein Rückgabewert kleiner oder gleich 32 zeigt einen Fehler an. Die möglichen Fehlerwerte finden Sie im folgenden Abschnitt »Fehler«.

Fehler Die Funktion *ShellExecute* gibt den Wert 31 zurück, wenn es keine Verknüpfung für den angegebenen Dateityp gibt oder wenn es keine Verknüpfung für die angegebene Aktion innerhalb des Dateityps gibt. Die anderen möglichen Fehlerwerte sind:

Wert	Bedeutung
0	Nicht genügend Speicher, Programmdatei war defekt oder Relozierungen waren ungültig.
2	Datei nicht gefunden.
3	Pfad nicht gefunden.
5	Es wurde versucht, eine Task dynamisch zu linken oder es ist ein Fehler bei der gemeinsamen Nutzung der Datei bzw. ein Netzschutz-Fehler aufgetreten.
6	Die Bibliothek benötigt separate Datensegmente für jede Task.
8	Nicht genügend Speicher, um die Anwendung zu starten.
10	Falsche Windows-Version.
11	Ausführbare Datei war ungültig. Entweder handelt es sich nicht um eine Windows-Anwendung oder der ausführbare Code hat einen Fehler.
12	Die Anwendung wurde für ein anderes Betriebssystem entworfen.
13	Die Anwendung wurde für MS-DOS 4.0 entworfen.
14	Unbekannter Typ der ausführbaren Datei.
15	Eine Real-Modus-Anwendung, die für eine frühere Version von Windows entworfen wurde, sollte geladen werden.
16	Es sollte eine zweite Instanz einer ausführbaren Datei geladen werden, die mehrere Datensegmente enthält, die nicht das Attribut Nur-Lesen besitzen.
19	Eine komprimierte ausführbare Datei sollte geladen werden. Die Datei muß entkomprimiert werden, bevor sie geladen werden kann.

Wert	Bedeutung
20	Die .DLL-Datei war ungültig. Eine der für diese Anwendung benötigten dynamischen Linkbibliotheken war defekt.
21	Die Anwendung benötigt die 32-Bit-Erweiterungen von Microsoft Windows.

Hinweis

Die im Parameter *lpszFile* angegebene Datei kann eine Dokumentdatei oder eine ausführbare Datei sein. Handelt es sich um eine Dokumentdatei, öffnet oder druckt diese Funktion die Datei in Abhängigkeit vom Wert des Parameters *lpszOp*. Liegt eine ausführbare Datei vor, wird sie von dieser Funktion geöffnet, auch wenn der String »print« im Parameter *lpszOp* angegeben ist.

Siehe auch

FindExecutable

ShellProc 3.1

Syntax

```
LRESULT CALLBACK ShellProc(code, wParam, lParam)

int code;       /* Botschaftsbearbeitungs-Flag      */
WPARAM wParam;  /* Flag der aktuellen Task          */
LPARAM lParam;  /* nicht definiert                  */
```

Beschreibung

Die Funktion *ShellProc* ist eine bibliotheksdefinierte Callback-Funktion, mit deren Hilfe eine Shell-Anwendung nützliche Systembenachrichtigungen empfangen kann.

Parameter

code: Bestimmt einen Shell-Benachrichtigungs-Code. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
HSHELL_ACTIVATESHELLWINDOW	Die Shell-Anwendung sollte ihr Hauptfenster aktivieren.
HSHELL_WINDOWCREATED	Ein nicht zugeordnetes Hauptfenster wurde erzeugt. Das Fenster existiert bereits, wenn das System eine *ShellProc*-Funktion aufruft.
HSHELL_WINDOWDESTROYED	Ein nicht zugeordnetes Hauptfenster soll freigegeben werden. Das Fenster existiert noch, wenn das System eine *ShellProc*-Funktion aufruft.

wParam: Bestimmt zusätzlich Informationen für die Shell-Anwendung. Die Interpretation dieses Parameters hängt vom Wert des Parameters *code* wie folgt ab:

code	*wParam*
HSHELL_ACTIVATE-SHELLWINDOW	Nicht verwendet.
HSHELL_WINDOW-CREATED	Bestimmt das Handle des zu erzeugenden Fensters.
HSHELL_WINDOW-DESTROYED	Bestimmt das Handle des freizugebenden Fensters.

lParam: Reserviert, wird nicht verwendet.

Rückgabewert Der Rückgabewert sollte Null sein.

Hinweis Anwendungen müssen diese Callback-Funktion installieren, indem sie den Filtertyp *WH_SHELL* und die Adresse der Prozedurinstanz der Callback-Funktion in einem Aufruf der Funktion *SetWindowsHook* angeben.

ShellProc ist ein Platzhalter für den bibliotheksdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Bibliothek exportiert werden.

Siehe auch *DefHookProc, SendMessage, SetWindowsHook*

SpoolFile 3.1

Syntax

```
HANDLE SpoolFile(lpszPrinter, lpszPort, lpszJob,
   lpszFile)

LPSTR lpszPrinter;   /* Druckername                */
LPSTR lpszPort;      /* Anschluß                   */
LPSTR lpszJob;       /* Name des Druckauftrags     */
LPSTR lpszFile;      /* Dateiname                  */
```

Beschreibung *SpoolFile* schreibt eine Datei in die Warteschlange. Diese Funktion wird normalerweise von Gerätetreibern verwendet.

Parameter *lpszPrinter:* Zeigt auf einen nullterminierten String, der den Namen des Druckers angibt, z.B. »HP LaserJet IIP«.

lpszPort: Zeigt auf einen nullterminierten String, der den Namen des Anschlusses bestimmt, z.B.»LPT1:«. Es muß sich um einen lokalen Anschluß handeln.

lpszJob: Zeigt auf einen nullterminierten String, der den Namen des Druckauftrags für das Spool-Programm bestimmt. Dieser String darf nicht länger als 32 Zeichen sein, einschließlich des abschließenden Null-Zeichens.

lpszFile: Zeigt auf einen nullterminierten String, der den Pfad und den Dateinamen der Datei angibt, die in die Warteschlan-

ge des Spool-Programms geschrieben werden soll. Diese Datei enthält Druckerdaten.

Rückgabewert Der Rückgabewert ist das globale Handle, das an das Spool-Programm übergeben wird, wenn die Funktion erfolgreich ist. Andernfalls kann es sich um einen der folgenden Fehlerwerte handeln:

SP_APPABORT
SP_ERROR
SP_NOTREPORTED
SP_OUTOFDISK
SP_OUTOFMEMORY
SP_USERABORT

Hinweis Anwendungen sollten sicherstellen, daß der Spooler vor dem Aufruf der Funktion *SpoolFile* aktiviert wurde.

StackTraceCSIPFirst 3.1

Syntax

```
#include <toolhelp.h>
BOOL StackTraceCSIPFirst(lpste, wSS, wCS, wIP, wBP)

STACKTRACEENTRY FAR* lpste; /* Adresse der Stack-
                               Frame-Struktur        */
WORD wSS;                  /* Wert des Registers SS  */
WORD sCS;                  /* Wert des Registers CS  */
WORD wIP;                  /* Wert des Registers IP  */
WORD wBP;                  /* Wert des Registers BP  */
```

Beschreibung *StackTraceCSIPFirst* füllt die angegebene Struktur mit Informationen, die den angegebenen Stack-Frame beschreiben

Parameter *lpste:* Zeigt auf eine *STACKTRACEENTRY*-Struktur, die Informationen über den Stack aufnehmen soll. Die *STACKTRACEENTRY*-Struktur hat folgende Form:

```
typedef struct tagSTACKTRACEENTRY {   /* ste */
    DWORD dwSize;
    HTASK hTask;
    WORD wSS;
    WORD wBP;
    WORD wCS;
    WORD wIP;
    HMODULE hModule;
    WORD wSegment;
    WORD wFlags;
} STACKTRACEENTRY;
```

wSS: Enthält den Wert des Registers *SS*. Dieser Wert wird zusammen mit *wBP* benutzt, um den nächsten Eintrag im Stack-Trace zu ermitteln.

wCS: Enthält den Wert des Registers *CS* des ersten Stack-Frame.

wIP: Enthält den Wert des Registers *IP* des ersten Stack-Frame.

wBP: Enthält den Wert des Registers *BP*. Dieser Wert wird zusammen mit *wSS* benutzt, um den nächsten Eintrag im Stack-Trace zu ermitteln.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *StackTraceFirst* kann zum Beginn einer Überprüfung des Stack jeder Task außer der aktuellen Task benutzt werden. Ist eine Task nicht aktiv, verwaltet der Kernel ihren Status, inklusive des aktuellen Stack, der Stack-Zeiger und der Werte in den Registern CS, IP und BP. Der Kernel verwaltet diese Werte nicht für die aktuelle Task. Aus diesem Grund muß für eine Überprüfung des Stack der aktuellen Task die Funktion *StackTraceCSIPFirst* benutzt werden. Anwendungen können die Überprüfung fortsetzen, indem sie die Funktion *StackTraceNext* aufrufen.

Anwendungen müssen die *STACKTRACEENTRY*-Struktur initialisieren und ihre Größe im Element *dwSize* in Byte angeben, bevor sie die Funktion *StackTraceCSIPFirst* aurufen.

Siehe auch *StackTraceNext*, *StackTraceFirst*

StackTraceFirst 3.1

Syntax

```
#include <toolhelp.h>
BOOL StackTraceFirst(lpste, htask)

STACKTRACEENTRY FAR* lpste;   /* Adresse der Stack-
                                 Frame-Struktur      */
HTASK htask;                  /* Handle der Task     */
```

Beschreibung Die Funktion *StackTraceFirst* füllt die angegebene Struktur mit Informationen, die den ersten Stack-Frame der angegebenen Task beschreiben.

Parameter *lpste:* Zeigt auf eine *STACKTRACEENTRY*-Struktur, die Informationen über den ersten Stack-Frame der Task aufnehmen soll. Die *STACKTRACEENTRY*-Struktur hat folgende Form:

```
typedef struct tagSTACKTRACEENTRY {   /* ste */
    DWORD dwSize;
    HTASK hTask;
    WORD wSS;
    WORD wBP;
    WORD wCS;
    WORD wIP;
    HMODULE hModule;
```

```
    WORD wSegment;
    WORD wFlags;
} STACKTRACEENTRY;
```

htask: Bezeichnet die Task, deren Stack beschrieben werden soll.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis *StackTraceFirst* kann zum Beginn einer Überprüfung des Stack jeder Task außer der aktuellen Task benutzt werden. Ist eine Task nicht aktiv, verwaltet der Kernel ihren Status, inklusive des aktuellen Stack, der Stack-Zeiger und der Werte in den Registern CS, IP und BP. Der Kernel verwaltet diese Werte nicht für die aktuelle Task. Aus diesem Grund muß für eine Überprüfung des Stack der aktuellen Task die Funktion *StackTraceCSIPFirst* benutzt werden. Anwendungen können die Überprüfung fortsetzen, indem sie die Funktion *StackTraceNext* aufrufen.

Anwendungen müssen die *STACKTRACEENTRY*-Struktur initialisieren und ihre Größe im Element *dwSize* in Byte angeben, bevor sie die Funktion *StackTraceFirst* aurufen.

Siehe auch *StackTraceCSIPFirst*, *StackTraceNext*

StackTraceNext 3.1

Syntax

```
#include <toolhelp.h>
BOOL StackTraceNext(lpste)

STACKTRACEENTRY FAR* lpste;   /* Adresse der Stack-
                                 Frame-Struktur      */
```

Beschreibung Die Funktion *StackTraceNext* füllt die angegebene Struktur mit Informationen, die den nächsten Stack-Frame bei einer Überprüfung des Stack beschreiben.

Parameter *lpste:* Zeigt auf eine *STACKTRACEENTRY*-Struktur, die Informationen über den nächsten Stack-Frame aufnehmen soll. Die *STACKTRACEENTRY*-Struktur hat folgende Form:

```
typedef struct tagSTACKTRACEENTRY {   /* ste */
    DWORD dwSize;
    HTASK hTask;
    WORD wSS;
    WORD wBP;
    WORD wCS;
    WORD wIP;
    HMODULE hModule;
    WORD wSegment;
    WORD wFlags;
} STACKTRACEENTRY;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *StackTraceNext* kann verwendet werden, um eine Überprüfung des Stack fortzusetzen, die mit den Funktionen *StackTraceFirst* oder *StackTraceCSIPFirst* begonnen wurde.

Siehe auch *StackTraceCSIPFirst*, *StackTraceFirst*, *STACKTRACEENTRY*

StartDoc 3.1

Syntax

```
int StartDoc(hdc, lpdi)

HDC hdc;             /* Handle des Gerätekontexts     */
DOCINFO FAR* lpdi;   /* Zeiger auf eine DOCINFO-
                        Struktur                      */
```

Beschreibung Die Funktion *StartDoc* startet einen Druckauftrag. Unter Windows 3.1 ersetzt diese Funktion den Drucker-Escape-Code *STARTDOC*.

Parameter *hdc:* Bezeichnet den Gerätekontext für den Druckauftrag.

lpdi: Zeigt auf eine *DOCINFO*-Struktur, die den Namen des Dokuments und den Namen der Ausgabedatei enthält. Die *DOCINFO*-Struktur hat folgende Form:

```
typedef struct {   /* di */
    int cbSize;
    LPCSTR lpszDocName;
    LPCSTR lpszOutput;
} DOCINFO;
```

Rückgabewert Wenn die Funktion erfolgreich ist, wird ein positiver Wert, andernfalls *SP_ERROR* zurückgegeben.

Hinweis Anwendungen sollten die Funktion *StartDoc* unmittelbar vor einem Druckauftrag aufrufen. Die Verwendung dieser Funktion stellt sicher, daß Dokumente mit mehr als einer Seite nicht von anderen Druckaufträgen unterbrochen werden.

Die Funktion *StartDoc* sollte nicht in Metafiles verwendet werden.

Siehe auch *EndDoc*, *Escape*

StartPage 3.1

Syntax

```
int StartPage(hdc)

HDC hdc;             /* Handle des Gerätekontexts     */
```

Beschreibung Die Funktion *StartPage* bereitet den Druckertreiber zur Aufnahme von Daten vor.

Parameter *hdc:* Bezeichnet den Gerätekontext für den Druckauftrag.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert größer als Null. Er ist kleiner oder gleich Null, wenn ein Fehler auftritt.

Hinweis Das System deaktiviert die Funktion *ResetDC* zwischen Aufrufen von *StartPage* und *EndPage*. Anwendungen können den Gerätemodus also nur zwischen den Seiten ändern.

Siehe auch *EndPage*, *Escape*, *ResetDC*

SubtractRect 3.1

Syntax

```
BOOL SubtractRect(lprcDest, lprcSource1, lprcSource2)

RECT FAR* lprcDest;          /* Zeiger auf das
                                Zielrechteck          */
const RECT FAR* lprcSource1; /* Zeiger auf das
                                Rechteck, von dem
                                abgezogen werden soll */
const RECT FAR* lprcSource2; /* Zeiger auf das Recht-
                                eck, das abgezogen
                                werden soll           */
```

Beschreibung Die Funktion *SubtractRect* ermittelt die Koordinaten eines Rechtecks, indem sie ein Rechteck von einem anderen subtrahiert.

Parameter *lprcDest:* Zeigt auf eine *RECT*-Struktur, die die Abmessungen des neuen Rechtecks aufnehmen soll. Die *RECT*-Struktur hat folgende Form:

```
typedef struct tagRECT {   /* rc */
    int left;
    int top;
    int right;
    int bottom;
} RECT;
```

lprcSource1: Zeigt auf eine *RECT*-Struktur, von der ein Rechteck subtrahiert werden soll.

lprcSource2: Zeigt auf die *RECT*-Struktur, die von dem durch den Parameter *lprcSource1* bezeichneten Rechteck abgezogen werden soll.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Das im Parameter *lprcSource2* angegebene Rechteck wird von dem in *lprcSource1* angegebenen Rechteck nur dann subtrahiert, wenn die angegebenen Rechtecke sich entweder in x- oder in y-Richtung vollständig überschneiden. Hat zum Bei-

spiel *lprcSource1* die Koordinaten (10, 10, 100, 100) und *lprcSource2* (50, 50, 150, 150), hat das durch *lprcDest* bezeichnete Rechteck die gleichen Koordinaten wie *lprcSource1*, sobald die Funktion zurückkehrt. Ist *lprcSource1* (10, 10, 100, 100) und *lprcSource2* (50, 10, 150, 150), enthält das durch *lprcDest* bezeichnete Rechteck die Koordinaten (10, 10, 50, 100), wenn die Funktion zurückkehrt.

Siehe auch *IntersectRect*, *UnionRect*

SysMsgProc 3.1

Syntax

```
LRESULT CALLBACK SysMsgProc(code, wParam, lParam)

int code;        /* Botschaftstyp                */
WPARAM wParam;   /* nicht definiert              */
LPARAM lParam;   /* Zeiger auf eine MSG-Struktur */
```

Beschreibung Die Funktion *SysMsgProc* ist eine bibliotheksdefinierte Callback-Funktion, die vom System aufgerufen wird, nachdem ein Dialogfenster, ein Meldungsfenster oder ein Menü eine Botschaft erhalten hat. Der Aufruf findet statt, bevor die Botschaft bearbeitet wird. Die Callback-Funktion kann die Botschaft für jede Anwendung im System bearbeiten oder verändern.

Parameter *code:* Bestimmt den Typ der zu bearbeitenden Botschaft. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
MSGF_DIALOGBOX	Botschaften innerhalb einer Dialogfenster- oder Meldungsfensterfunktion werden bearbeitet.
MSGF_MENU	Tastatur- und Mausbotschaften in einem Menü werden bearbeitet.

Wenn der Parameter *code* kleiner als Null ist, muß die Callback-Funktion die Botschaft ohne weitere Bearbeitung an *CallNextHookEx* übergeben. Sie sollte dann den Rückgabewert von *CallNextHookEx* zurückgeben.

wParam: Muß *NULL* sein.

lParam: Zeigt auf die *MSG*-Struktur, die die Botschaft enthält. Die *MSG*-Struktur hat folgende Form:

```
typedef struct tagMSG {   /* msg */
    HWND hwnd;
    UINT message;
    WPARAM wParam;
    LPARAM lParam;
```

```
    DWORD time;
    POINT pt;
} MSG;
```

Rückgabewert Der Rückgabewert sollte ungleich Null sein, wenn die Funktion die Botschaft bearbeitet. Andernfalls sollte er Null sein.

Hinweis Diese Callback-Funktion muß sich in einer dynamischen Linkbibliothek (DLL) befinden.

Anwendungen müssen diese Callback-Funktion installieren, indem sie den Filtertyp *WH_SYSMSGFILTER* und die Adresse der Prozedurinstanz der Callback-Funktion in einem Aufruf der Funktion *SetWindowsHookEx* angeben.

SysMsgProc ist ein Platzhalter für den bibliotheksdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Bibliothek exportiert werden.

Siehe auch *CallNextHookEx, MessageBox, SetWindowsHookEx*

SystemHeapInfo 3.1

Syntax

```
#include <toolhelp.h>
BOOL SystemHeapInfo(lpshi)

SYSHEAPINFO FAR* lpshi;   /* Adresse der Struktur für
                             die Heap-Informationen   */
```

Beschreibung Die Funktion *SystemHeapInfo* füllt die angegebenen Struktur mit Informationen, die den Heap der Programme USER.EXE und GDI.EXE beschreiben.

Parameter *lpshi:* Zeigt auf eine *SYSHEAPINFO*-Struktur, die Informationen über den USER- und den GDI-Heap aufnimmt. Die *SYSHEAPINFO*-Struktur hat folgende Form:

```
typedef struct tagSYSHEAPINFO {   /* shi */
    DWORD dwSize;
    WORD wUserFreePercent;
    WORD wIFreePercent;
    HGLOBAL hUserSegment;
    HGLOBAL hGDISegment;
} SYSHEADINFO;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion wurde für Überwachungszwecke bereitgestellt. Anwendungen müssen die *SYSHEAPINFO*-Struktur initialisieren und ihre Größe im Element *dwSize* in Byte angeben, bevor sie *SystemHeapInfo* aufrufen.

SystemParametersInfo 3.1

Syntax

```
BOOL SystemParametersInfo(uAction, uParam, lpvParam,
   fuWinIni)

UINT uAction;          /* abzufragende oder zu
                          setzende Systemparameter    */
UINT uParam;           /* abhängig von den System-
                          parametern                  */
void FAR* lpvParam;    /* abhängig von den
                          Systemparametern            */
UINT fuWinIni;         /* Aktualisierungs-Flag für
                          WIN.INI                     */
```

Beschreibung

Die Funktion *SystemParametersInfo* fragt Systemeinstellungen ab oder setzt sie. Die Funktion kann in Abhängigkeit vom Parameter *fuWinIni* auch die Datei WIN.INI aktualisieren, während ein Systemparameter neu gesetzt wird.

uAction: Bestimmt die Systemeinstellung, die abgefragt oder gesetzt werden soll.

Möglich sind folgende Werte:

Wert	Bedeutung
SPI_GETBEEP	Ruft einen *BOOL*-Wert ab. Dieser zeigt an, ob die akustische Warnmeldung an- oder abgeschaltet ist.
SPI_GETBORDER	Ruft den Multiplizierungsfaktor für Rahmen ab. Dieser bestimmt die Breite von Fensterrahmen.
SPI_GETFASTTASK-SWITCH	Überprüft, ob die schnelle Task-Umschaltung aktiviert ist.
SPI_GETGRID-GRANULARITY	Ermittelt den aktuellen Wert der Punktdichte des Desktop-Rechtecks für die Einstellung von Fenstergrößen.
SPI_GETICONTITLE-LOGFONT	Ruft die virtuellen Schriftinformationen der aktuellen Schrift für Symbolbeschriftungen ab.
SPI_GETICON-TITLEWRAP	Ermittelt, ob der Umbruch für Symbolbeschriftungen an- oder ausgeschaltet ist.
SPI_GETKEYBOARD-DELAY	Ruft die Einstellung der Wiederholverzögerung für die Tastatur ab.
SPI_GETKEYBOARD-SPEED	Ruft die Einstellung der Wiederholrate für die Tastatur ab.

Wert	Bedeutung
SPI_GETMENU-DROPALIGNEMENT	Ermittelt, ob Popup-Menüs linksbündig oder rechtsbündig zum entsprechenden Element der Hauptmenüleiste ausgerichtet werden.
SPI_GETMOUSE	Ruft die Mausgeschwindigkeit sowie die x- und y-Schwellenwerte der Maus ab. Diese Werte verwendet Windows, um die Beschleunigung der Mausbewegungen zu berechnen.
SPI_GETSCREEN-SAVEACTIVE	Ruft einen *BOOL*-Wert ab. Dieser zeigt an, ob der Bildschirmschoner an- oder ausgeschaltet ist.
SPI_GETSCREEN-SAVETIMOUT	Ruft den Timeout-Wert des Bildschirmschoners ab.
SPI_ICONHORIZON-TALSPACING	Setzt die Breite einer Symbolzelle in Pixel.
SPI_ICONVERTI-CALSPACING	Setzt die Höhe einer Symbolzelle in Pixel.
SPI_LANGDRIVER	Fordert den Anwender zum Laden eines neuen Sprachtreibers auf.
SPI_SETBEEP	Schaltet die akustische Warnmeldung ein oder aus.
SPI_SETBORDER	Setzt den Multiplizierungsfaktor für Rahmen. Dieser bestimmt die Breite von Fensterrahmen.
SPI_SETDESK-PATTERN	Setzt das aktuelle Desktop-Muster auf den Wert, der im Eintrag »Pattern=« in der Datei WIN.INI steht oder auf den durch *lpvParam* bezeichneten Wert.
SPI_SETDESK-WALLPAPER	Bestimmt den Namen der Datei, die das Bitmap für den Desktop-Hintergrund enthält.
SPI_SETDOUBLE-CLKHEIGHT	Setzt die Höhe des Rechtecks, innerhalb dessen der zweite Mausklick stattfinden muß, damit er als Doppelklick interpretiert wird.
SPI_SETDOUBLE-CLICKTIME	Setzt die Doppelklickzeit für die Maus. Die Doppelklickzeit ist die maximale Anzahl von Millisekunden, die zwischen dem ersten und dem zweiten Mausklick eines Doppelklicks liegen darf.

Wert	Bedeutung
SPI_SETDOUBLE-CLKWIDTH	Setzt die Breite des Rechtecks, innerhalb dessen der zweite Mausklick stattfinden muß, damit er als Doppelklick interpretiert wird.
SPI_SETFASTTASK-SWITCH	Aktiviert oder deaktiviert die schnelle Task-Umschaltung.
SPI_SETGRID-GRANULARITY	Setzt den Wert der Punktdichte des Desktop-Rechtecks für die Einstellung von Fenstergrößen.
SET_SETICONTITLE-LOGFONT	Setzt die Schrift für die Beschriftungen von Symbolen.
SPI_SETICON-TITLEWRAP	Schaltet den Umbruch für Symbolbeschriftungen an oder aus.
SPI_SETKEYBOARD-DELAY	Setzt die Wiederholverzögerung für die Tastatur.
SPI_SETKEYBOARD-SPEED	Setzt die Wiederholrate für die Tastatur.
SPI_SETMENUDROP-ALIGNEMENT	Setzt den Ausrichtungswert für Popup-Menüs.
SPI_SETMOUSE	Setzt die Mausgeschwindigkeit sowie die x- und y-Schwellenwerte der Maus.
SPI_SETMOUSE-BUTTONSWAP	Tauscht die Funktion der rechten und linken Maustaste aus oder stellt den Ausgangszustand wieder her.
SPI_SETSCREEN-SAVEACTIVE	Setzt den Status des Bildschirmschoners.
SPI_SETSCREEN-SAVETIMEOUT	Setzt den Timeout-Wert des Bildschirmschoners.

uParam: Dieser Parameter hängt vom Parameter *uAction* ab. Detaillierte Informationen finden Sie im folgenden »Hinweis«.

lpvParam: Dieser Parameter hängt vom Parameter *uAction* ab. Detaillierte Informationen finden Sie im folgenden »Hinweis«.

fuWinIni: Wird eine Systemeinstellung verändert, legt dieser Parameter fest, ob die Datei WIN.INI entsprechend aktualisiert wird. Wird WIN.INI aktualisiert, empfängt jedes Hauptfenster eine Botschaft *WM_WININICHANGE*. Der Parameter *fWinIni* kann einen der folgenden Werte annehmen:

Wert	Bedeutung
SPIF_UPDATEINI-FILE	Schreibt die neuen Werte der Systemeinstellungen in die Datei WIN.INI.
SPIF_SENDWIN-INICHANGE	Übergibt die Botschaft *WM_WININI-CHANGE* nach Aktualisierung der Datei WIN.INI.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich war. Er ist Null, wenn ein Fehler auftritt.

Hinweis Diese Funktion ist für Anwendungen wie die Systemsteuerung vorgesehen, die dem Anwender die Anpassung der Windows-Umgebung ermöglichen.

Die folgende Tabelle beschreibt die Parameter *uParam* und *lpvParam* für jede der *SPI_*-Konstanten:

Konstante	Wirkung von *wParam* und *lpvParam*
SPI_GETBEEP	*wParam:* 0 *lpvParam:* Zeigt auf eine *BOOL*-Variable, die *TRUE* ist, wenn die akustische Warnmeldung eingeschaltet ist. Ist die Warnmeldung ausgeschaltet, hat die Variable den Wert *FALSE*.
SPI_GETBORDER	*wParam:* 0 *lpvParam:* Zeigt auf eine Integer-Variable, die den Multiplizierungsfaktor für Rahmen aufnimmt.
SPI_GETFAST-TASKSWITCH	*wParam:* 0 *lpvParam:* Zeigt auf eine *BOOL*-Variable. Diese ist *TRUE*, wenn die schnelle Task-Umschaltung aktiviert, *FALSE*, wenn sie deaktiviert ist.
SPI_GETGRID-GRANULARITY	*wParam:* 0 *lpvParam:* Zeigt auf eine Integer-Variable, die den Wert für die Punktdichte des Desktop-Rechtecks für die Einstellung von Fenstergrößen aufnimmt.
SPI_GETICON-TITLELOGFONT	*wParam:* Bestimmt die Größe einer *LOG-FONT*-Struktur. *lpvParam:* Zeigt auf eine *LOGFONT*-Struktur, die die virtuellen Schriftinformationen aufnimmt.

Konstante	Wirkung von *wParam* und *lpvParam*
SPI_GETICON-TITLEWRAP	*wParam:* 0 *lpvParam:* Zeigt auf eine *BOOL*-Variable, die *TRUE* ist, wenn der Umbruch eingeschaltet wurde. Andernfalls ist sie *FALSE*.
SPI_GETKEYBOARD-DELAY	*wParam:* 0 *lpvParam:* Zeigt auf eine Integer-Variable, die die Einstellung der Wiederholverzögerung für die Tastatur aufnimmt.
SPI_GETKEYBOARD-SPEED	*wParam:* 0 *lpvParam:* Zeigt auf eine *WORD*-Variable, die die Einstellung der Wiederholrate der Tastatur aufnimmt.
SPI_GETMENU-DROPALIGNEMENT	*wParam:* 0 *lpvParam:* Zeigt auf eine *BOOL*-Variable, die *FALSE* enthält, wenn Popup-Menüs linksbündig ausgerichtet werden. Bei rechtsbündiger Ausrichtung enthält sie *TRUE*.
SPI_GETMOUSE	*wParam:* 0 *lpvParam:* Zeigt auf ein Integer-Array mit dem Namen *lpiMouse*. *lpiMouse*[*0*] erhält den Wert des WIN.INI-Eintrags für *MouseThreshold1*, *lpiMouse*[*1*] erhält den Wert des Eintrags für *MouseThreshold2* und *lpiMouse*[*2*] erhält den Wert von *MouseSpeed*.
SPI_GETSCREEN-SAVEACTIVE	*wParam:* 0 *lpvParam:* Zeigt auf eine *BOOL*-Variable, die *TRUE* enthält, wenn der Bildschirmschoner aktiviert ist, andernfalls enthält sie *FALSE*.
SPI_GETSCREEN-SAVETIMEOUT	*wParam:* 0 *lpvParam:* Zeigt auf eine Integer-Variable, die den Timeout-Wert des Bildschirmschoners in Sekunden aufnimmt.
SPI_ICON-HORIZONTAL-SPACING	*wParam:* Die Breite des Abstands zwischen Symbolen in Pixel. *lpvParam: NULL*, wenn die Breite der Symbolzelle in *uParam* zurückgegeben wird. Handelt es sich bei diesem Wert um einen Zeiger auf einen Integer, wird der horizontale Symbolabstand in dieser Variablen zurückgegeben und *uParam* ignoriert.

Konstante	Wirkung von *wParam* und *lpvParam*
SPI_ICON-VERTICAL-SPACING	*wParam:* Die Höhe des Abstands zwischen Symbolen in Pixel. *lpvParam: NULL*, wenn die Höhe der Symbolzelle in *uParam* zurückgegeben wird. Handelt es sich bei diesem Wert um einen Zeiger auf einen Integer, wird der vertikale Symbolabstand in dieser Variablen zurückgegeben und *uParam* ignoriert.
SPI_LANGDRIVER	*wParam:* 0 *lpvParam:* Zeigt auf einen String, der den Dateinamen des neuen Sprachtreibers enthält. Anwendungen sollten beim Wechseln des Sprachtreibers sicherstellen, daß die übrigen länderspezifischen Einstellungen konsistent bleiben.
SPI_SETBEEP	*wParam: TRUE* aktiviert die akustische Warnmeldung, *FALSE* schaltet sie aus. *lpvParam: NULL*
SPI_SETBORDER	*wParam:* Der Multiplizierungsfaktor für Rahmen. *lpvParam: NULL*
SPI_SETDESK-PATTERN	*wParam:* 0 oder -1 *lpvParam:* Bestimmt das Desktop-Muster. Ist dieser Wert *NULL* und der Parameter *uParam* -1, wird der Wert erneut aus der Datei WIN.INI gelesen. Bei diesem Wert kann es sich auch um einen nullterminierten String (*LPSTR*) handeln, der eine Folge von acht Zahlen enthält. Diese repräsentieren das Muster. »170 85 170 85 170 85 170 85« steht z.B. für eine Grauschraffur mit 50% Schwärzung.
SPI_SETDESK-WALLPAPER	*wParam:* 0 *lpvParam:* Zeigt auf einen String, der den Namen der Bitmap-Datei bestimmt.
SPI_SETDOUBLE-CLKHEIGHT	*wParam:* Die Höhe des Doppelklickbereichs in Pixel. *lpvParam: NULL*
SPI_SETDOUBLE-CLICKTIME	*wParam:* Doppelklickzeit in Millisekunden. *lpvParam: NULL*

Konstante	Wirkung von *wParam* und *lpvParam*
SPI_SETDOUBLE-CLKWIDTH	*wParam:* Die Breite des Doppelklickbereichs in Pixel.
	lpvParam: NULL
SPI_SETFAST-TASKSWITCH	*wParam: TRUE* aktiviert die schnelle Task-Umschaltung, *FALSE* deaktiviert sie.
	lpvParam: NULL
SPI_SETGRID-GRANULARITY	*wParam:* Punktdichte des Desktop-Rechtecks.
	lpvParam: NULL
SPI_SETICON-TITLELOGFONT	*wParam:* Die Größe der*LOGFONT*-Struktur.
	lpvParam: Zeigt auf eine *LOGFONT*-Struktur, die die Schrift für die Beschriftungen von Symbolen definiert. Wenn *uParam* auf Null und *lParam* auf *NULL* gesetzt ist, verwendet Windows die Schrift und die Abstände, die beim Start von Windows gültig waren.
SPI_SETICON-TITLEWRAP	*wParam: TRUE* schaltet den Umbruch ein, *FALSE* schaltet ihn aus.
	lpvParam: NULL
SPI_SETKEYBOARD-DELAY	*wParam:* Bestimmt die Einstellung der Wiederholverzögerung der Tastatur.
	lpvParam: NULL
SPI_SETKEYBOARD-SPEED	*wParam:* Bestimmt die Einstellung der Wiederholrate der Tastatur.
	lpvParam: NULL
SPI_SETMENUDROP-ALIGNEMENT	*wParam: TRUE* bestimmt rechtsbündige, *FALSE* linksbündige Ausrichtung.
	lpvParam: NULL
SPI_SETMOUSE	*wParam:* 0
	lpvParam: Zeigt auf ein Integer-Array mit dem Namen *lpiMouse*. *lpiMouse*[*0*] erhält den Wert des WIN.INI-Eintrags für *MouseThreshold1*, *lpiMouse*[*1*] erhält den Wert des Eintrags für *MouseThreshold2* und *lpiMouse*[*2*] erhält den Wert von *MouseSpeed*.
SPI_SETMOUSE-BUTTONSWAP	*wParam: TRUE* vertauscht die Funktionen der linken und der rechten Maustaste, *FALSE* stellt die ursprüngliche Belegung wieder her.
	lpvParam: NULL

Konstante	Wirkung von *wParam* und *lpvParam*
SPI_SETSCREEN-SAVEACTIVE	*wParam: TRUE* aktiviert den Bildschirmschoner, *FALSE* deaktiviert ihn.
	lpvParam: NULL
SPI_SETSCREEN-SAVETIMEOUT	*wParam:* Setzt die Dauer bis zur Bildschirmabschaltung in Sekunden.
	lpvParam: NULL

Beispiel

Das folgende Beispiel ruft den Wert des Eintrags *DoubleClick-Speed* aus der Datei WIN.INI ab und verwendet ihn zur Initialisierung eines Editierfelds. In diesem Beispiel wird der anwenderspezifische Wert während der Bearbeitung einer *WM_COMMAND*-Botschaft aus dem Editierfeld abgerufen und zum Setzen der Doppelklickzeit verwendet.

```
char szBuf[32];
int iResult;
case WM_INITDIALOG:

    /* Editierfeld mit der aktuellen Doppelklickzeit
       initialisieren. */

    iResult = GetProfileInt("windows",
                            "DoubleClickSpeed", 550);
    itoa(iResult, szBuf, 10);
    SendDlgItemMessage(hdlg, IDD_DLCKTIME,
       WM_SETTEXT, 0, (DWORD) (LPSTR) szBuf);

    .
    . /* Andere Steuerelemente initialisieren */
    .

    return FALSE;

case WM_COMMAND:
    switch(wParam) {

    case IDOK:

        /* Doppelklickzeit mit benutzerdefiniertem Wert
           initialisieren. */

        SendDlgItemMessage(hdlg, IDD_DLCKTIME,
           WM_GETTEXT, sizeof(szBuf),
           (DWORD) (LPSTR) szBuf);
      SystemParametersInfo(SPI_SETDOUBLECLICKTIME,
        atoi(szBuf), (LPVOID) NULL, SPIF_UPDATEINIFILE |
        SPIF_SENDWININICHANGE);
      .
      . */ Andere Systemparameter setzen. */
      .
```

```
        EndDialog(hdlg, TRUE);
        return TRUE;
    }
```

TaskFindHandle 3.1

Syntax

```
#include <toolhelp.h>
BOOL TaskFindHandle(lpte, htask)

TASKENTRY FAR* lpte;   /* Adresse einer TASKENTRY-
                          Struktur                  */
HTASK htask;           /* Handle der Task           */
```

Beschreibung Die Funktion *TaskFindHandle* füllt die angegebene Struktur mit Informationen, die die angebene Task beschreiben.

Parameter *lpte:* Zeigt auf eine *TASKENTRY*-Struktur, die Informationen über die Task aufnehmen soll. Die *TASKENTRY*-Struktur hat folgende Form:

```
typedef struct tagTASKENTRY {   /* te */
    DWORD dwSize;
    HTASK hTask;
    HTASK hTaskParent;
    HINSTANCE hInst;
    HMODULE hModule;
    WORD wSS;
    WORD wSP;
    WORD wStackTop;
    WORD wStackMinimum;
    WORD wStackBottom;
    WORD wcEvents;
    HGLOBAL hQueue;
    char szModule[MAX_MODULE_NAME + 1];
    WORD wPSPOffset;
    HANDLE hNext;
} TASKENTRY;
```

htask: Bezeichnet die zu beschreibende Task.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *TaskFindHandle* kann benutzt werden, um eine Überprüfung der Warteschlange der Tasks zu beginnen. Anwendungen können weitere Einträge in diese Liste untersuchen, indem sie die Funktion *TaskNext* aufrufen.

Anwendungen müssen die *TASKENTRY*-Struktur initialisieren und ihre Größe im Element *dwSize* in Byte angeben, bevor sie *TaskFindHandle* aufrufen.

Siehe auch *TaskFirst*, *TaskNext*

TaskFirst 3.1

Syntax

```
#include <toolhelp.h>
BOOL TaskFirst(lpte)

TASKENTRY FAR* lpte;            /* Adresse der
                                   TASKENTRY-Struktur */
```

Beschreibung Die Funktion *TaskFirst* füllt die angegebene Struktur mit Informationen über die erste Task in der Warteschlange der Tasks.

Parameter *lpte:* Zeigt auf eine *TASKENTRY*-Struktur, die Informationen über die erste Task aufnehmen soll. Die *TASKENTRY*-Struktur hat folgende Form:

```
typedef struct tagTASKENTRY {   /* te */
    DWORD dwSize;
    HTASK hTask;
    HTASK hTaskParent;
    HINSTANCE hInst;
    HMODULE hModule;
    WORD wSS;
    WORD wSP;
    WORD wStackTop;
    WORD wStackMinimum;
    WORD wStackBottom;
    WORD wcEvents;
    HGLOBAL hQueue;
    char szModule[MAX_MODULE_NAME + 1];
    WORD wPSPOffset;
    HANDLE hNext;
} TASKENTRY;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *TaskFirst* kann verwendet werden, um eine Überprüfung der Warteschlange der Tasks zu beginnen. Anwendungen können die folgenden Einträge in der Warteschlange mit Hilfe der Funktion *TaskNext* untersuchen.

Anwendungen müssen die *TASKENTRY*-Struktur initialisieren und ihre Größe im Element *dwSize* in Byte angeben, bevor sie *TaskFirst* aufrufen.

Siehe auch *TaskFindHandle, TaskNext*

TaskGetCSIP 3.1

Syntax

```
#include <toolhelp.h>
DWORD TaskGetCSIP(htask)

HTASK htask;            /* Handle der Task        */
```

Beschreibung Die Funktion *TaskGetCSIP* gibt den nächsten CS:IP-Wert einer ruhenden Task zurück. Diese Funktion ist hilfreich, wenn Anwendungen die Startadresse der Ausführung einer ruhenden Task benötigen.

Parameter *htask:* Bezeichnet die Task, deren CS:IP-Wert ermittelt wird. *TaskGetCSIP* kann nur auf ruhende Tasks angewendet werden.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert der nächste CS:IP-Wert. Ist der Parameter *htask* ungültig, wird *NULL* zurückgegeben.

Hinweis *TaskGetCSIP* sollte nicht aufgerufen werden, wenn *htask* die aktuelle Task bezeichnet.

Siehe auch *DirectedYield, TaskSetCSIP, TaskSwitch*

TaskNext 3.1

Syntax

```
#include <toolhelp.h>
BOOL TaskNext(lpte)

TASKENTRY FAR* lpte;      /* Adresse der TASKENTRY-
                             Struktur                */
```

Beschreibung Die Funktion *TaskNext* füllt die angegebene Struktur mit Informationen über die nächste Task in der Warteschlange der Tasks.

Parameter *lpte:* Zeigt auf eine *TASKENTRY*-Struktur, die Informationen über die nächste Task aufnimmt. Die *TASKENTRY*-Struktur hat folgende Form:

```
#include <toolhelp.h>

typedef struct tagTASKENTRY { /* te */
  DWORD dwSize;
  HTASK hTask;
  HTASK hTaskParent;
  HINSTANCE hInst;
  HMODULE hModule;
  WORD wSS;
  WORD wSP;
  WORD wStackTop;
  WORD wStackMinimum;
  WORD wStackBootom;
  WORD wcEvents;
  HGLOBAL hQueue;
  char szModule[MAX_MODULE_NAME + 1];
  WORD wPSPOffset;
  HANDLE hNext;
} TASKENTRY;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Mit *TaskNext* kann eine Überprüfung der Warteschlange der Tasks fortgesetzt werden. Die Überprüfung muß mit der Funktion *TaskFirst* oder *TaskFindHandle* begonnen werden.

Siehe auch *TaskFindHandle, TaskFirst*

TaskSetCSIP 3.1

Syntax

```
#include <toolhelp.h>
DWORD TaskSetCSIP(htask, wCS, wIP)

HTASK htask;            /* Handle der Task          */
WORD wCS;               /* Wert im CS-Register      */
WORD wIP;               /* Wert im IP-Register      */
```

Beschreibung Die Funktion *TaskSetCSIP* setzt den CS:IP-Wert einer ruhenden Task. Wird die Steuerung an diese Task übergeben, beginnt die Ausführung an der angegebenen Adresse.

Parameter *htask:* Bezeichnet die Task, der ein neuer CS:IP-Wert zugewiesen werden soll.

wCS: Enthält den neuen Wert des Registers CS.

wIP: Enthält den neuen Wert des Registers IP.

Rückgabewert Der Rückgabewert ist der vorherige CS:IP-Wert dieser Task. Die Funktion *TaskSwitch* verwendet diesen Wert. Der Rückgabewert ist *NULL*, wenn der Parameter *htask* ungültig ist.

Hinweis *TaskSetCSIP* sollte nicht aufgerufen werden, wenn *htask* die aktuelle Task bezeichnet.

Siehe auch *DirectedYield, TaskGetCSIP, TaskSwitch*

TaskSwitch 3.1

Syntax

```
#include <toolhelp.h>
BOOL TaskSwitch(htask, dwNewCSIP)

HTASK htask;          /* Handle der Task             */
DWORD dwNewCSIP;      /* Ausführungsadresse
                         innerhalb der Task           */
```

Beschreibung Die Funktion *TaskSwitch* schaltet auf die angegebene Task um. Die Task beginnt die Ausführung an der angegebenen Adresse.

Parameter *htask:* Bezeichnet die neue Task.

dwNewCSIP: Bezeichnet die Adresse innerhalb der angegebenen Task, an der die Ausführung begonnen werden soll. Diese

Adresse darf sich keinesfalls innerhalb eines Codesegments der angegebenen Task befinden.

Rückgabewert Wenn die Task-Umschaltung erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Sobald die durch den Parameter *htask* bezeichnete Task die Steuerung abgibt, kehrt *TaskSwitch* zur aufrufenden Anwendung zurück.

TaskSwitch ändert den CS:IP-Wert des Stack-Frame der Task auf den im Parameter *dwNewCSIP* angegebenen Wert und ruft anschließend die Funktion *DirectedYield* auf.

Siehe auch *DirectedYield, TaskSetCSIP, TaskGetCSIP*

TerminateApp 3.1

Syntax

```
#include <toolhelp.h>
void TerminateApp(htask, wFlags)

HTASK htask;                /* Handle der Task          */
WORD wFlags;                /* Beendigungs-Flags        */
```

Beschreibung Die Funktion *TerminateApp* beendet die angegebene Anwendungsinstanz (Task).

Parameter *htask:* Bezeichnet die zu beendende Task. Wenn dieser Parameter *NULL* ist, bezeichnet er die aktuelle Task.

wFlags: Bestimmt die Art der Beendigung der Task. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
UAE_BOX	Ruft den Windows-Kernel auf, um das Meldungsfenster *Fehler im Anwendungsprogramm* anzuzeigen und beendet dann die Task.
NO_UAE_BOX	Ruft den Windows-Kernel auf, um die Task zu beenden. Das Meldungsfenster *Fehler im Anwendungsprogramm* wird nicht dargestellt. In diesem Fall sollte die Interrupt- oder Benachrichtigungs-Callback-Funktion der Anwendung eine Fehlermeldung oder eine Warnung darstellen.

Rückgabewert Diese Funktion gibt nur dann einen Wert zurück, wenn *htask* ungleich *NULL* ist und nicht die aktuelle Task bezeichnet.

Hinweis Die Funktion *TerminateApp* hebt die Registrierung aller Callback-Funktionen auf, die mit den ToolHelp-Bibliotheksfunk-

tionen registriert wurden. Anschließend wird die Anwendung beendet, als ob die angegebene Task einen GP-Fehler (General Protection) oder einen anderen Fehler erzeugt hätte.

TerminateApp sollte nur in Testanwendungen benutzt werden, da die Funktion möglicherweise nicht alle Objekte freigibt, die der beendeten Anwendung zugeordnet sind.

Siehe auch *InterruptRegister, InterruptUnRegister, NotifyRegister, NotifyUnRegister*

TimerCount 3.1

Syntax

```
#include <toolhelp.h>
BOOL TimerCount(lpti)

TIMERINFO FAR* lpti;   /* Adresse der Struktur für
                          die Ausführungszeiten      */
```

Beschreibung Die Funktion *TimerCount* füllt die angegebene Struktur mit den Ausführungszeiten der aktuellen Task und der VM (Virtual Machine).

Parameter *lpti:* Zeigt auf eine *TIMERINFO*-Struktur, die die Ausführungszeiten aufnimmt.

Die *TIMERINFO*-Struktur hat folgende Form:

```
#include <toolhelp.h>

typedef struct tagTIMERINFO { /* ti */
  DWORD dwSize;
  DWORD dwmsSinceStart;
  DWORD dwmsThisVMM;
} TIMERINFO;
```

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *TimerCount* ermöglicht die millisekundengenaue Ermittlung von Zeitangaben. Im erweiterten 386-Modus verwendet *TimerCount* das VTD (virtual timer device), um die genauen Ausführungszeiten zu bestimmen.

Im Standardmodus ruft *TimerCount* die Funktion *GetTickCount* auf, die ihrerseits Informationen zurückgibt, die bis auf einen Zeittakt (ca. 55 Millisekunden) genau sind. Anschließend liest *TimerCount* die Hardware-Uhr, um die Differenz in Millisekunden bis zum nächsten Zeittakt zu berechnen. Das Ergebnis ist bis auf eine Millisekunde genau.

Anwendungen müssen die *TIMERINFO*-Struktur initialisieren und ihre Größe im Element *dwSize* in Byte angeben, bevor sie *TimerCount* aufrufen.

Siehe auch *GetTickCount*

TimerProc 2.x

Syntax

```
void CALLBACK TimerProc(hwnd, msg, idTimer, dwTime)

HWND hwnd;       /* Handle des Fensters, das die
                    Timer-Botschaften erhalten soll   */
UINT msg;        /* WM_TIMER-Botschaft                */
UINT idTimer;    /* Bezeichner des Timer              */
DWORD dwTime;    /* aktuelle Systemzeit               */
```

Beschreibung Die Funktion *TimerProc* ist eine anwendungsdefinierte Callback-Funktion, die *WM_TIMER*-Botschaften bearbeitet.

Parameter *hwnd:* Bezeichnet das Fenster, das dem Timer zugeordnet ist.

msg: Bestimmt die *WM_TIMER*-Botschaft.

idTimer: Enthält den Bezeichner des Timer.

dwTime: Bestimmt die aktuelle Systemzeit.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Hinweis *TimerProc* ist ein Platzhalter für den anwendungsdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Anwendung exportiert werden.

Siehe auch *KillTimer, SetTimer*

UnAllocDiskSpace 3.1

Syntax

```
#include <stress.h>
void UnAllocDiskSpace(drive)

UINT drive;      /* Plattenpartition                  */
```

Beschreibung Die Funktion *UnAllocDiskSpace* löscht die Datei STRESS.EAT aus dem Hauptverzeichnis des angegebenen Laufwerks. Dadurch wird der Plattenspeicher freigegeben, der zuvor durch die Funktion *AllocDiskSpace* belegt wurde.

Parameter *drive:* Bestimmt die Plattenpartition, von der die Datei STRESS.EAT gelöscht werden soll. Möglich ist einer der folgenden Werte:

Wert	Bedeutung
EDS_WIN	Löscht die Datei auf der Windows-Partition.
EDS_CUR	Löscht die Datei auf der aktuellen Partition.
EDS_TEMP	Löscht die Datei auf der Partition, die das in TEMP angegebene Verzeichnis enthält.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Siehe auch *AllocDiskSpace*

UnAllocFileHandles 3.1

Syntax
```
#include <stress.h>
void UnAllocFileHandles(void)
```

Beschreibung Die Funktion *UnAllocFileHandles* gibt alle Datei-Handles frei, die mit der Funktion *AllocFileHandles* reserviert wurden.

Parameter Diese Funktion hat keine Parameter.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Siehe auch *AllocFileHandles*

UndeleteFile 3.1

Syntax
```
#include <wfext.h>
int FAR PASCAL UndeleteFile(hwndParent, lpszDir)

HWND hwndParent;     /* Handle des Fensters des
                        Datei-Managers              */
LPSTR lpszDir;       /* Adresse des Namens des
                        anfänglichen Verzeichnisses */
```

Beschreibung Die Funktion *UndeleteFile* ist eine anwendungsdefinierte Callback-Funktion. Sie wird vom Datei-Manager aufgerufen, wenn der Anwender den Befehl *Wiederherstellen* aus dem Menü *Datei* des Datei-Managers auswählt.

Parameter *hwndParent:* Bezeichnet das Datei-Manager-Fenster. Dynamische Linkbibliotheken für *Wiederherstellen* sollten dieses Handle benutzen, um das übergeordnete Fenster für Dialogfenster oder Meldungsfenster zu bestimmen.

lpszDir: Zeigt auf einen nullterminierten String, der den Namen des anfänglichen Verzeichnisses enthält.

Rückgabewert Der Rückgabewert ist einer der folgenden Werte, wenn die Funktion erfolgreich ist:

Wert	Bedeutung
-1	Ein Fehler ist aufgetreten.
IDOK	Eine Datei wurde wiederhergestellt. Der Datei-Manager zeichnet seine Fenster neu.
IDCANCEL	Es wurde keine Datei wiederhergestellt.

UnhookWindowsHookEx 3.1

Syntax

```
BOOL UnhookWindowsHookEx(hhook)

HHOOK hhook;          /* Handle der zu entfernenden
                         Filterfunktion              */
```

Beschreibung Die Funktion *UnhookWindowsHookEx* entfernt eine anwendungsdefinierte Filterfunktion aus einer Kette von Filterfunktionen. Filterfunktionen bearbeiten Ereignisse, bevor diese an die Botschaftsschleife der Anwendung innerhalb der Funktion *WinMain* übergeben werden.

Parameter *hhook:* Bezeichnet die Filterfunktion, die entfernt werden soll. Dieser Wert wird von der Funktion *SetWindowsHookEx* zurückgegeben, wenn der Filter installiert wird.

Rückgabewert Wenn die Funktion erfolgreich ist, ist der Rückgabewert ungleich Null, andernfalls Null.

Hinweis Die Funktion *UnhookWindowsHookEx* muß immer in Kombination mit der Funktion *SetWindowsHookEx* benutzt werden.

Beispiel Das folgende Beispiel verwendet die Funktion *UnhookWindowsHookEx,* um einen Botschaftsfilter für kontextsensitive Hilfe innerhalb eines Dialogfensters zu entfernen:

```
DLGPROC  lpfnAboutProc;
HOOKPROC lpfnFilterProc;
HHOOK    hhook;

case IDM_ABOUT:
   lpfnAboutProc = (DLGPROC) MakeProcInstance
                   (About, hinst);
   lpfnFilterProc = (HOOKPROC) MakeProcInstance
                    (FilterFunc, hinst);
   hhook = SetWindowsHookEx (WH_MSGFILTER,
           lpfnFilterProc, hinst, (HTASK) NULL);
   DialogBox (hinst, "AboutBox", hwnd, lpfnAboutProc);
   UnhookWindowsHookEx (hhook);
   FreeProcInstance ((FARPROC) lpfnFilterProc);
   FreeProcInstance ((FARPROC) lpfnAboutProc);
   break;
```

Siehe auch *CallNextHookEx, SetWindowsHookEx*

VerFindFile 3.1

Syntax

```
#include <ver.h>
UINT VerFindFile(flags, lpszFilename, lpszWinDir,
   lpszAppDir, lpszCurDir, lpuCurDirLen, lpszDestDir,
   lpuDestDirLen)
```

```
UINT flags;               /* Flags der Quelldatei    */
LPCSTR lpszFilename;      /* Adresse des Puffers für
                            die Datei               */
LPCSTR lpszWinDir;        /* Adresse des Windows-
                            Verzeichnisses          */
LPCSTR lpszAppDir;        /* Adresse des Anwendungs-
                            verzeichnisses          */
LPSTR lpszCurDir;         /* Adresse des Puffers für
                            das aktuelle Verzeichnis*/
UINT FAR* lpuCurDirLen;   /* Adresse der Puffergröße
                            für das Verzeichnis     */
LPSTR lpszDestDir;        /* Adresse des Puffers für
                            das Zielverzeichnis     */
UINT FAR* lpuDestDirLen;  /* Adresse der Größe des
                            Zielverzeichnisses      */
```

Beschreibung

Die Funktion *VerFindFile* ermittelt, wo eine Datei installiert werden soll. Dies ist davon abhängig, ob eine andere Version der Datei im System existiert. Die von *VerFindFile* zurückgegebenen Werte werden in nachfolgenden Aufrufen der Funktion *VerInstallFile* verwendet.

Parameter

flags: Enthält einen Bitmaske mit Flags. Ist dieser Parameter *VFFF_ISSHAREDFILE,* können mehrere Anwendungen gleichzeitig auf diese Datei zugreifen. *VerFindFile* legt mit Hilfe dieser Information fest, wohin die Datei kopiert werden sollte. Alle anderen möglichen Werte sind für zukünftige Versionen dieser Funktion vorgesehen.

lpszFilename: Zeigt auf einen nullterminierten String, der den Namen der zu installierenden Datei bestimmt. Dieser Name sollte nur den Dateinamen und die Namenserweiterung, nicht aber den Pfad umfassen.

lpszWinDir: Zeigt auf einen nullterminierten String, der das Windows-Verzeichnis bestimmt. Dieser String wird von der Funktion *GetWindowsDir* zurückgegeben. Die DLL-Version (dynamische Linkbibliothek) von *VerFindFile* ignoriert diesen Parameter.

lpszAppDir: Zeigt auf einen String, der den Laufwerksnamen und das Verzeichnis bestimmt, in dem das Installationsprogramm eine Reihe zusammengehöriger Dateien installiert. Wenn das Installationsprogramm eine Anwendung installiert, wird dieses das Anwendungsverzeichnis. Solange kein anderes Verzeichnis angegeben wird, ist dies auch das Arbeitsverzeichnis der jeweiligen Anwendung.

lpszCurDir: Zeigt auf einen Puffer, der den Pfad der aktuellen Version der zu installierenden Datei aufnimmt. Dieser Pfadname ist ein nullterminierter String. Ist keine aktuelle Version

der Datei installiert, enthält der Puffer das Quellverzeichnis der zu installierenden Datei. Der Puffer muß mindestens *_MAX_PATH* Byte groß sein.

lpuCurDirLen: Zeigt auf einen nullterminierten String, der die Länge des durch *lpszCurDir* bezeichneten Puffers in Byte enthält. Bei Rückkehr der Funktion enthält *lpuCurDirLen* die Größe der in *lpszCurDir* zurückgegebenen Daten in Byte. Dabei wird das abschließende Nullzeichen mitgezählt. Wenn der Puffer zu klein für die zurückgegebenen Daten ist, ist *lpuCurDirLen* größer als die tatsächliche Puffergröße.

lpszDestDir: Zeigt auf einen Puffer, der den Pfad zu dem von *VerFindFile* empfohlenen Installationsverzeichnis enthält. Der Pfad ist ein nullterminierter String. Der Puffer muß mindestens *_MAX_PATH* Byte groß sein.

lpuDestDirLen: Zeigt auf die Länge (in Byte) des durch *lpszDestDir* bezeichneten Puffers. Bei Rückkehr der Funktion enthält *lpuDestDirLen* die Größe der in *lpszDestDir* zurückgegebenen Daten in Byte. Das abschließende Nullzeichen wird dabei mitgezählt. Wenn der Puffer zu klein für die zurückgegebenen Daten ist, übersteigt *lpuDestDirLen* die tatsächliche Größe des Puffers.

Rückgabewert

Bei erfolgreicher Ausführung der Funktion ist der Rückgabewert eine Bitmaske, die den Dateistatus kennzeichnet. Möglich sind einer oder mehrere der folgenden Werte:

Wert	Bedeutung
VFF_CURNEDEST	Zeigt an, daß sich die aktuell installierte Version der Datei nicht im empfohlenen Verzeichnis befindet.
VFF_FILEINUSE	Zeigt an, daß Windows die aktuell installierte Dateiversion benutzt. Aus diesem Grund kann die Datei weder überschrieben noch gelöscht werden.
VFF_BUFFTOSMALL	Zeigt an, daß zumindest ein Puffer zu klein ist, um den entsprechenden String aufzunehmen. Anwendungen sollten die Parameter *lpuCurDirLen* und *lpuDestDirLen* überprüfen, um den betreffenden Puffer zu ermitteln.

Alle anderen Werte sind für zukünftige Nutzungen reserviert.

Hinweis

Die DLL-Version von *VerFindFile* sucht mit Hilfe der Funktion *OpenFile* nach einer Kopie der angegebenen Datei. Die .LIB-

Version dieser Funktion durchsucht das Windows-Verzeichnis, das Systemverzeichnis und die Verzeichnisse in der DOS-Variablen PATH nach der Datei.

VerFindFile ermittelt das Systemverzeichnis über das angegebene Windows-Verzeichnis oder mit Hilfe der DOS-Variablen PATH.

Wenn der Parameter *wFlags* anzeigt, daß die Datei exklusiv zu einer einzigen Anwendung gehört – also nicht *VFFF_ISSHAREDFILE* ist –, schlägt *VerFindFile* die Installation der Datei im Anwendungsverzeichnis vor. Wird eine gemeinsam benutzte Kopie von Windows ausgeführt, empfiehlt die Funktion die Installation im Windows-Verzeichnis. Wird eine private Kopie von Windows ausgeführt, empfiehlt diese Funktion die Installation im Systemverzeichnis.

Siehe auch *VerInstallFile*

VerInstallFile 3.1

Syntax

```
#include <ver.h>
DWORD VerInstallFile(flags, lpszSrcFilename,
   lpszDestFilename, lpszSrcDir, lpszDestDir,
   lpszCurDir, lpszTmpFile, lpwTmpFileLen)

UINT flags;              /* Quelldatei-Flags          */
LPCSTR lpszSrcFilename;  /* Adresse des Puffers für
                            den Quelldateinamen       */
LPCSTR lpszDestFilename; /* Adresse des Puffers für
                            den Zieldateinamen        */
LPCSTR lpszSrcDir;       /* Adresse des Puffers für
                            den Quellverzeichnis-
                            namen                     */
LPCSTR lpszDestDir;      /* Adresse des Puffers für
                            den Zielverzeichnis-
                            namen                     */
LPCSTR lpszCurDir;       /* Adresse des Puffers für
                            das bisherige Verzeichnis */
LPSTR lpszTmpFile;       /* Adresse des Puffers für
                            den Namen der temporären
                            Datei                     */
UINT FAR* lpwTmpFileLen; /* Adresse des Puffers für
                            die Größe der temporären
                            Datei                     */
```

Beschreibung Die Funktion *VerInstallFile* versucht eine Datei mit Hilfe der Informationen zu installieren, die die Funktion *VerFindFile* zurückgibt. *VerInstallFile* entkomprimiert die Datei mit Hilfe der Funktion *LZCopy* und prüft auf Fehler (zum Beispiel veraltete Datei).

Parameter

flags: Enthält eine Bitmaske mit Flags. Dieser Parameter kann eine Kombination der folgenden Werte enthalten:

Wert	Bedeutung
VIFF_FORCE-INSTALL	Installiert die Datei ohne Berücksichtigung der Versionsnummer. Die Funktion prüft während der Installation nur auf physikalische Fehler. Wenn *flags VIFF_FORCEINSTALL* enthält und *lpszTmpFileLen* kein Zeiger auf Null ist, übergeht *VerInstallFile* die Versionsüberprüfung für die temporäre und die Zieldatei. Außerdem wird die temporäre Datei mit dem durch *lpszSrcFilename* angegebenen Dateinamen umbenannt, wenn sie im Zielverzeichnis existiert, die Zieldatei nicht benutzt wird und der Anwender die nötigen Zugriffsrechte besitzt, um die Zieldatei zu löschen und die temporäre Datei umzubenennen. Der Rückgabewert von *VerInstallFile* sollte auf Fehler geprüft werden.
VIFF_DONT-DELETEOLD	Installiert die Datei, ohne die ältere Version zu löschen, wenn sich letztere nicht im Zielverzeichnis befindet. Befindet sich die ältere Version im Zielverzeichnis, wird sie bei erfolgreicher Installation von *VerInstallFile* durch die neue Version ersetzt.

Alle anderen Werte sind für eine zukünftige Nutzung reserviert.

lpszSrcFilename: Zeigt auf den Namen der zu installierenden Datei. Dabei handelt es sich um den Dateinamen im Verzeichnis, auf das *lpszSrcDir* zeigt. Der Dateiname sollte also nur den eigentlichen Dateinamen und die Namenserweiterung enthalten, nicht den Pfad. *VerInstallFile* öffnet die Quelldatei durch Aufruf der Funktion *LZOpenFile*. Es können also sowohl die angegebenen Dateien als auch komprimierte Dateien, die mit der Option */r* von COMPRESS.EXE umbenannt wurden, bearbeitet werden.

lpszDestFilename: Zeigt auf den Namen, den *VerInstallFile* der Datei während der Installation zuordnet. Dieser Name kann von dem Dateinamen in dem durch *lpszSrcFilename* bezeichneten Verzeichnis abweichen. Der neue Dateiname sollte nur den eigentlichen Dateinamen und die Namenserweiterung, nicht aber den Pfad enthalten.

lpszSrcDir: Zeigt auf den Puffer, der den Namen des Verzeichnisses enthält, in dem sich die neue Datei befindet.

lpszDestDir: Zeigt auf den Puffer, der den Namen des Verzeichnisses enthält, in dem die neue Dateiversion installiert werden soll. Die Funktion *VerFindFile* gibt diesen Wert im Parameter *lpszDestDir* zurück.

lpszCurDir: Zeigt auf den Puffer, der den Namen des Verzeichnisses enthält, in dem die ältere Version der Datei existiert. *VerFindFile* gibt diesen Wert im Parameter *lpszCurDir* zurück. Wenn *flags* nicht *VIFF_DONTDELETEOLD* enthält und der in *lpszDestFilename* angegebene Dateiname bereits im Verzeichnis *lpszCurDir* existiert, wird die bereits existierende Dateiversion gelöscht. Ist *lpszCurDir* ein Zeiger auf *NULL*, existiert keine frühere Version der Datei im System.

lpszTempFile: Zeigt auf einen Puffer, der beim Aufruf von *VerInstallFile* leer sein sollte. Die Funktion füllt diesen Puffer mit dem Namen einer temporären Kopie der Quelldatei. Der Puffer muß mindestens *_MAX_PATH* Byte groß sein.

lpwTmpFileLen: Zeigt auf die Größe des durch *lpszTmpFile* bezeichneten Puffers. Bei der Rückkehr enthält *lpwTmpFileLen* den Umfang der in *lpszTmpFile* zurückgegebenen Daten. Das abschließende Nullzeichen wird dabei mitgezählt. Wenn der Puffer zu klein ist, um alle Daten aufzunehmen, ist *lpwTmpFileLen* größer als die tatsächliche Puffergröße. Wenn *flags* *VIFF_FORCEINSTALL* enthält und *lpwTmpFileLen* kein Zeiger auf Null ist, benennt *VerInstallFile* die temporäre Datei mit dem durch *lpszSrcFilename* angegebenen Namen um.

Rückgabewert

Bei erfolgreicher Ausführung der Funktion ist der Rückgabewert eine Bitmaske, die Besonderheiten anzeigt. Möglich sind einer oder mehrere der folgenden Werte:

Wert	Bedeutung
VIF_TEMPFILE	Zeigt an, daß sich die temporäre Kopie der neuen Datei im Zielverzeichnis befindet. Der Grund für Fehler wird mit anderen Flags angezeigt. Anwendungen sollten nötigenfalls die temporäre Datei löschen, wenn dieses Flag gesetzt ist.
VIF_MISMATCH	Zeigt an, daß die neue und die alte Datei sich in einem oder mehreren Attributen unterscheiden. Dieser Fehler kann durch einen erneuten Aufruf von *VerInstallFile* mit dem Flag *VIFF_FORCEINSTALL* umgangen werden.
VIF_SRCOLD	Zeigt an, daß die zu installierende Datei älter als die bereits vorhandene ist. Dieser Fehler kann durch einen erneuten Aufruf von *VerInstallFile* mit dem Flag *VIFF_FORCEINSTALL* umgangen werden.

Wert	Bedeutung
VIF_DIFFLANG	Zeigt an, daß die neue und die bereits vorhandene Datei verschiedene Sprach- oder Code-Seiten-Werte haben. Dieser Fehler kann durch erneuten Aufruf von *VerInstallFile* mit dem Flag *VIFF_FORCEINSTALL* umgangen werden.
VIF_DIFF-CODEPG	Zeigt an, daß die neue Datei eine Code-Seite benötigt, die von der aktuell laufenden Windows-Version nicht dargestellt werden kann. Dieser Fehler kann durch erneuten Aufruf von *VerInstallFile* mit dem Flag *VIFF_FORCEINSTALL* umgangen werden.
VIF_DIFFTYPE	Zeigt an, daß die neue Datei einen anderen Typ oder Untertyp besitzt oder ein anderes Betriebssystem benutzt, als die bereits existierende Version. Dieser Fehler kann durch erneuten Aufruf von *VerInstallFile* mit dem Flag *VIFF_FORCEINSTALL* umgangen werden.
VIF_WRITEPROT	Zeigt an, daß die bereits vorhandene Dateiversion schreibgeschützt ist. Das Installationsprogramm sollte das Nur-Lesen-Bit der Zieldatei zurücksetzen, bevor die Installation fortgesetzt wird.
VIF_FILEINUSE	Zeigt an, daß die bereits vorhandene Datei von Windows benutzt wird und deshalb nicht gelöscht werden kann.
VIF_OUTOFSPACE	Zeigt an, daß die Funktion keine temporäre Datei erzeugen kann, da im Ziellaufwerk nicht genügend Speicherplatz vorhanden ist.
VIF_ACCESSVIOLATION	Zeigt an, daß eine Erstellungs-, Löschen- oder Umbenennungsoperation wegen einer Verletzung der Zugriffsrechte fehlgeschlagen ist.
VIF_SHARINGVIOLATION	Zeigt an, daß eine Erstellungs-, Löschen- oder Umbenennungsoperation wegen einer Verletzung der gemeinsamen Nutzungsrechte fehlgeschlagen ist.
VIF_CANNOTCREATE	Zeigt an, daß die Funktion keine temporäre Datei erzeugen kann. Der genaue Grund wird möglicherweise durch andere Flags beschrieben.
VIF_CANNOTDELETE	Zeigt an, daß die Funktion die Zieldatei oder die existierende Version der Datei in einem anderen Verzeichnis nicht löschen kann.

Wert	Bedeutung
	Ist das Flag *VIF_TEMPFILE* gesetzt, so ist die Installation fehlgeschlagen und die Zieldatei kann wahrscheinlich nicht gelöscht werden.
VIF_CANNOT-RENAME	Zeigt an, daß die Funktion die temporäre Datei nicht umbenennen kann, die Zieldatei aber bereits gelöscht wurde.
VIF_OUTOF-MEMORY	Zeigt an, daß die Funktion die Operation wegen mangelnder Speicherkapazität nicht vollständig durchführen konnte. Dies tritt meistens bei der Entkomprimierung einer komprimierten Datei auf.
VIF_CANNOT-READSRC	Zeigt an, daß die Funktion die Quelldatei nicht lesen kann. Dies kann daran liegen, daß der Pfadname fehlerhaft ist, die Datei nicht existiert oder die komprimierte Datei beschädigt wurde. Verwenden Sie *LZOpenFile*, um diese Fehlerbedingungen zu überprüfen. Die Funktion *OpenFile* kann hierzu nicht benutzt werden, da sie die Namen von komprimierten Dateien nicht richtig übersetzt. *VIF_CANNOT-READSRC* führt nicht zum Setzen der Bits *VIF_ACCESSVIOLATION* oder *VIF_SHARING-VIOLATION*.
VIF_CANNOT-READDST	Zeigt an, daß die Funktion die existierenden Zieldateien nicht lesen kann. Deshalb können die Dateiattribute nicht untersucht werden.
VIF_BUFFTO-SMALL	Zeigt an, daß der Puffer *lpszTmpFile* zu klein für die Aufnahme des Namens der temporären Quelldatei war. Bei der Rückkehr enthält *lpwTmpFileLen* die benötigte Puffergröße.

Alle anderen Werte sind für eine zukünftige Nutzung reserviert.

Hinweis

VerInstallFile wurde für die Verwendung in Installationsprogrammen entworfen. Sie kopiert die durch *lpszSrcFilename* angegebene Datei von der Installationsdiskette in eine temporäre Datei im Zielverzeichnis. Nötigenfalls entkomprimiert *VerInstallFile* die Datei mit Hilfe der Funktionen in LZEXPAND.DLL.

Existiert im Zielverzeichnis bereits eine Kopie der Datei, vergleicht *VerInstallFile* die Versionsinformationen der temporären Datei mit denen der existierenden Datei. Ist die bereits existierende Datei jüngeren Datums als die temporäre Datei

oder zeigen sich deutliche Unterschiede in den Dateiattributen, gibt *VerInstallFile* einen oder mehrere Fehlerwerte zurück. Dateien mit unterschiedlichen Sprachen führen zum Beispiel zur Rückgabe des Werts *VIF_DIFFLANG* durch *VerInstallFile*.

VerInstallFile beläßt die temporäre Datei im Zielverzeichnis. Sind alle Fehler behebbar, können sie vom Installationsprogramm durch erneuten Aufruf von *VerInstallFile* mit dem Flag *VIFF_FORCEINSTALL* umgangen werden. In diesem Fall sollte *lpszSrcFilename* auf den Namen der temporären Datei zeigen. *VerInstallFile* löscht dann die bereits existierende Datei und benennt die temporäre Datei mit dem durch *lpszSrcFilename* angegebenen Dateinamen um. Zeigt das Bit *VIF_TEMPFILE* an, daß eine temporäre Datei existiert, muß die Anwendung diese Datei löschen, wenn die Installation nicht mit dem Flag *VIFF_FORCEINSTALL* erzwungen wird.

Versucht eine Anwendung, die Installation einer Datei nach einem nicht behebbaren Fehler (zum Beispiel *VIF_CANNOTREAADSRC*) zu erzwingen, wird die Datei von *VerInstallFile* nicht installiert.

Siehe auch *VerFindFile*

VerLanguageName 3.1

Syntax

```
#include <ver.h>
VerLanguageName(uLang, lpszLang, cbLang)

UINT uLang;       /* Sprachbezeichner von Microsoft   */
LPSTR lpszLang;   /* Adresse des Puffers für den
                     Sprach-String                    */
UINT cbLang;      /* Größe des Puffers                */
```

Beschreibung Die Funktion *VerLanguageName* wandelt den angegebenen binären Microsoft-Sprachbezeichner in einen String um, der die Sprache benennt.

Parameter *uLang:* Gibt den binären Microsoft-Sprachbezeichner an. So übersetzt *VerLanguageName* »0x040A« in »Kastilisches Spanisch«. Wenn *VerLanguageName* den Bezeichner nicht erkennt, zeigt *lpszLang* auf einen voreingestellten String: »Unbekannte Sprache«.

Eine vollständige Liste der von Windows unterstützten Sprachbezeichner finden Sie im »Hinweis«.

lpszLang: Zeigt auf den Puffer, der den nullterminierten String als Repräsentation der durch *uLang* bezeichneten Sprache aufnehmen soll.

nSize: Zeigt die Größe des Puffers, der durch *lpszLang* bezeichnet wird, in Byte an.

Rückgabewert Wenn die Funktion erfolgreich ist, gibt sie die Länge des String zurück, der den Sprachbezeichner repräsentiert. Dieser Wert berücksichtigt das abschließende Nullzeichen des String nicht. Ist der Wert größer als *cbLang*, wurde der String auf *cbLang* gekürzt. Der Rückgabewert ist Null, wenn ein Fehler auftritt. Unbekannte Werte für *uLang* erzeugen keinen Fehler.

Hinweis Normalerweise verwenden installierte Anwendungen diese Funktion, um den von der Funktion *VerQueryValue* zurückgegebenen Sprachbezeichner zu übersetzen. Der String kann z.B. in Benutzerabfragen bei konfligierenden Sprachen verwendet werden. Windows unterstützt folgende Sprachbezeichner:

Wert	Sprache
0x0401	Arabisch
0x0402	Bulgarisch
0x0403	Katalanisch
0x0404	Traditionelles Chinesisch
0x0405	Tschechisch
0x0406	Dänisch
0x0407	Deutsch
0x0408	Griechisch
0x0409	Amerikanisches Englisch
0x040A	Kastilisches Spanisch
0x040B	Finnisch
0x040C	Französisch
0x040D	Hebräisch
0x040E	Ungarisch
0x040F	Isländisch
0x0410	Italienisch
0x0411	Japanisch
0x0412	Koreanisch
0x0413	Niederländisch
0x0414	Norwegisch – Bokmal
0x0415	Polnisch
0x0416	Brasilianisches Portugiesisch
0x0417	Rätoromanisch
0x0418	Rumänisch
0x0419	Russisch
0x041A	Kroatoserbisch (lateinisch)
0x041B	Slowenisch
0x041C	Albanisch
0x041D	Schwedisch
0x041E	Thai
0x041F	Türkisch
0x0420	Urdu
0x0421	Bahasa
0x0804	Vereinfachtes Chinesisch

Wert	Sprache
0x0807	Schweizerdeutsch
0x0809	Britisches Englisch
0x080A	Mexikanisches Spanisch
0x080C	Belgisches Französisch
0x0810	Schweizerisches Italienisch
0x0813	Belgisches Niederländisch
0x0814	Norwegisch – Nynorsk
0x0816	Portugiesisch
0x081A	Serbokroatisch (kyrillisch)
0x0C1C	Kanadisches Französisch
0x100C	Schweizerisches Französisch

VerQueryValue 3.1

Syntax

```
#include <ver.h>
BOOL VerQueryValue(lpvBlock, lpszSubBlock, lplpBuffer,
   lpcb)

const void FAR* lpvBlock;   /* Adresse des Puffers für
                               die Versionsressource  */
LPCSTR lpszSubBlock;        /* Adresse des
                               abzurufenden Werts     */
VOID FAR* FAR* lplpBuffer;  /* Adresse des Puffers
                               für den Versionszeiger */
UINT FAR* lpcb;             /* Adresse des Puffers
                               für die Größe des
                               Versionswerts          */
```

Beschreibung *VerQueryValue* gibt ausgewählte Informationen aus der angegebenen Ressource mit Versionsinformationen zurück. Um die gewünschte Ressource zu ermitteln, muß die Funktion *GetFileVersionInfo* vor *VerQueryValue* aufgerufen werden.

Parameter *lpvBlock:* Zeigt auf den Puffer, der die Ressource mit den Versionsinformationen enthält, die von der Funktion *GetFileVersionInfo* zurückgegeben wurde.

lpszSubBlock: Zeigt auf einen nullterminierten String, der den abzurufenden Versionsinformationswert bestimmt. Der String besteht aus Namen, die jeweils durch einen Backslash (\) getrennt sind. Er kann folgende Form haben:

Form	Beschreibung
\	Bestimmt den Root-Block. Die Funktion ruft einen Zeiger auf die *VS_FIXEDFILEINFO*-Struktur für die Versionsinformationsressource ab.

Form	Beschreibung
\VarFileinfo\ Translation	Bestimmt die Übersetzungstabelle im Block mit den variablen Informationen. Die Funktion ruft einen Zeiger auf ein Array mit Bezeichnern der Sprache und des Zeichensatzes ab. Anwendungen verwenden diese Bezeichner, um den Namen eines sprachspezifischen Blocks in der Versionsinformationsressource zu erzeugen.
\StringFileInfo *sprachspezifischer Zeichensatz*\ *String-Name*	Bestimmt einen Wert in einem sprachspezifischen Block. Der Name *sprachspezifischer Zeichensatz* ist eine Verknüpfung der Bezeichner für die Sprache und für den Zeichensatz. Er ist in der Übersetzungstabelle der Ressource gespeichert. Der Name muß als hexadezimaler String angegeben werden. Der Name *String-Name* ist einer der vordefinierten Werte, die im »Hinweis« beschrieben sind.

lplpBuffer: Zeigt auf einen Puffer, der den Zeiger auf den Versionsinformationswert aufnimmt.

lpcb: Zeigt auf einen Puffer, der die Länge des Versionsinformationswerts in Byte aufnimmt.

Rückgabewert

Der Rückgabewert ist ungleich Null, wenn der angegebene Block existiert und Versionsinformationen verfügbar sind. Ist *lpcb* Null, ist kein Wert für den angegebenen Versionsinformationsnamen verfügbar.

Der Rückgabewert ist Null, wenn der angegebene Name nicht existiert oder die durch den Parameter *lpvBlock* bezeichnete Ressource ungültig ist.

Hinweis

String-Name im Parameter *lpszSubBlock* kann einen der folgenden vordefinierten Namen annehmen:

Name	Wert
Comments	Bestimmt zusätzliche Informationen, die zu Diagnosezwecken angezeigt werden sollten.
CompanyName	Bestimmt die Firma, die die Datei hergestellt hat, beispielsweise »Borland« oder »Borland International GmbH«. Dieser String muß angegeben werden.
FileDescription	Bestimmt eine Dateibeschreibung, die dem Benutzer dargestellt werden soll.

Name	Wert
	Dieser String kann in einer Liste angezeigt werden, wenn der Anwender Dateien für die Installation auswählt. Möglich ist z.B. »Tastaturtreiber für AT-Tastaturen« oder »Turbo C++ für Windows«. Dieser String muß angegeben werden.
FileVersion	Bestimmt die Versionsnummer der Datei, z.B. »3.10« oder »5.00.RC2«. Dieser String muß angegeben werden.
InternalName	Bestimmt den internen Namen der Datei, sofern ein solcher existiert. Dabei kann es sich z.B. um einen Modulnamen handeln, wenn die Datei eine dynamische Linkbibliothek ist. Besitzt die Datei keinen internen Namen, sollte dieser String den ursprünglichen Dateinamen ohne Namenserweiterung enthalten. Dieser String muß angegeben werden.
LegalCopyright	Bestimmt alle Copyright-Hinweise für diese Datei. Dies schließt den vollständigen Text des Hinweises, die entsprechenden Symbole und Daten ein. Möglich ist z.B. die Angabe »Copyright© Borland International 1990 - 1991«. Dieser String ist optional.
LegalTrademarks	Bestimmt alle Warenzeichen und registrierten Warenzeichen der Datei. Dies schließt den vollständigen Text aller Hinweise, Symbole, Nummern usw. ein. Möglich ist z.B. »Windows(TM) ist ein Warenzeichen der Microsoft Corporation«. Dieser String ist optional.
OriginalFilename	Bestimmt den ursprünglichen Namen der Datei, nicht aber ihren Pfad. So kann eine Anwendung feststellen, ob die Datei durch den Benutzer umbenannt wurde. Das Format des Namens hängt vom Dateisystem ab, für das die Datei erzeugt wurde. Dieser String muß angegeben werden.
PrivateBuild	Bestimmt Informationen über eine private Version der Datei. Möglich ist z.B. »Erstellt von TESTER1 auf \TESTBED«. Dieser String sollte nur verwendet werden, wenn das Flag *VS_FF_PRIVATEBUILD* im Element *dwFileFlags* der *VS_FIXEDFILEINFO*-Struktur des Root-Blocks angegeben ist.

Name	Wert
ProductName	Bestimmt den Namen des Produkts, mit dem die Datei ausgeliefert wird. Möglich ist z.B. »Microsoft Windows«. Dieser String muß angegeben werden.
ProductVersion	Bestimmt die Version des Produkts, mit dem die Datei ausgeliefert wird. Möglich ist z.B. »3.10« oder »5.00.RC2«. Dieser String muß angegeben werden.
SpecialBuild	Bestimmt, wie sich diese Version der Datei von der Standardversion unterscheidet. Möglich ist z.B. »Eigene Anpassung von TESTER1, mit der Mausprobleme auf den Computern M250 und M250E gelöst werden«. Dieser String sollte nur angegeben werden, wenn das Flag *VS_FF_SPECIALBUILD* im Element *dwFileFlags* der *VS_FIXEDFILEINFO*-Struktur im Root-Block gesetzt ist.

Beispiel

Das folgende Beispiel lädt die die Versionsinformationen einer dynamischen Linkbibliothek und ruft den Firmennamen (*CompanyName*) ab:

```
BYTE   abData[512];
DWORD  handle;
DWORD  dwSize;
LPBYTE lpBuffer;
char   szName[512];

dwSize = GetFileVersionInfoSize("c:\\dll\\sample.dll",
                                &handle));

GetFileVersionInfo("c:\\dll\\sample.dll",
                   handle, dwSize, abData));

VerQueryValue(abData, "\\VarFileInfo\\Translation",
              &lpBuffer, &dwSize));

if (dwSize!=0) {
   wsprintf(szName, "\\StringFileInfo\\" +
                    "%   81x\\CompanyName", &lpBuffer);
   VerQueryValue(abData, szName, &lpBuffer, &dwSize);

}
```

Siehe auch

GetFileVersionInfo

WindowProc 3.1

Syntax

```
LRESULT CALLBACK WindowProc(hwnd, msg wParam, lParam)

HWND hwnd;          /* Handle des Fensters          */
UINT msg;           /* Botschaft                    */
WPARAM wParam;      /* erster Botschaftsparameter   */
LPARAM lParam;      /* zweiter Botschaftsparameter  */
```

Beschreibung Die Funktion *WindowProc* ist eine anwendungsdefinierte Callback-Funktion, die Botschaften bearbeitet, die an ein Fenster übergeben werden.

Parameter *hwnd:* Bezeichnet das Fenster.

msg: Bestimmt die Botschaft.

wParam: Bestimmt 16 Bits zusätzlicher botschaftsabhängiger Informationen.

lParam: Bestimmt 32 Bits zusätzlicher botschaftsabhängiger Informationen.

Rückgabewert Der Rückgabewert ist das Ergebnis der Botschaftsbearbeitung. Der Wert hängt von der bearbeiteten Botschaft ab.

Hinweis *WindowProc* ist ein Platzhalter für den anwendungsdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Anwendung exportiert werden.

Siehe auch *DefWindowProc, RegisterClass*

WNetAddConnection 2.x

Syntax

```
UINT WNetAddConnection(lpszNetPath, lpszPassword,
   lpszLocalName

LPSTR lpszNetPath;    /* Adresse des Netzgeräts       */
LPSTR lpszPassword;   /* Adresse des Paßworts         */
LPSTR lpszLocalName;  /* Adresse des lokalen Geräts   */
```

Beschreibung Die Funktion *WNetAddConnection* leitet das angegebene lokale Gerät – Laufwerk oder Drucker – auf das angegebene gemeinsam genutzte Gerät oder den Remote-Server um.

Parameter *lpszNetPath:* Zeigt auf einen nullterminierten String, der das gemeinsam genutzte Gerät oder den Remote-Server bestimmt.

lpszPassword: Zeigt auf einen nullterminierten String, der das Netzpaßwort für das angegebene Gerät oder den Remote-Server bestimmt.

lpszLocalName: Zeigt auf einen nullterminierten String, der das lokale Laufwerk oder Gerät bestimmt, das umgeleitet werden

soll. Alle Strings für *lpszLocalName* (wie z.B. »LPT1«) können klein oder groß geschrieben werden. Nur die Gerätenamen A bis Z und LPT1 bis LPT3 werden benutzt.

Rückgabewert Einer der folgenden Werte wird zurückgegeben:

Wert	Bedeutung
WN_SUCCESS	Die Funktion war erfolgreich.
WN_NOT_-SUPPORTED	Die Funktion wurde nicht unterstützt.
WN_OUT_OF_-MEMORY	Nicht genügend Speicherplatz im System.
WN_NET_ERROR	Ein Netzfehler ist aufgetreten.
WN_BAD_POINTER	Der Zeiger war ungültig.
WN_BAD_-NETNAME	Der Ressourcenname des Netzes war ungültig.
WN_BAD_-LOCALNAME	Der Name des lokalen Geräts war ungültig.
WN_BAD_PASS-WORD	Das Paßwort war ungültig.
WN_ACCESS_-DENIED	Der Zugriff wurde verweigert.
WN_ALREADY_-CONNECTED	Das lokale Gerät war bereits mit einer Remote-Ressource verbunden.

Siehe auch *WNetCancelConnection, WNetGetConnection*

WNetCancelConnection 3.1

Syntax

```
UINT WNetCancelConnection(lpszName, fForce)

LPSTR lpszName;       /* Adresse des Geräts oder
                         der Ressource              */
BOOL fForce;          /* Schließen-Flag             */
```

Beschreibung Die Funktion *WNetCancelConnection* unterbricht die Anbindung an ein Netz.

Parameter *lpszName:* Zeigt entweder auf den Namen des umgeleiteten lokalen Geräts (z.B. »LPT1«) oder auf einen vollen Netzpfad. Ist ein Netzpfad angegeben, unterbricht der Treiber alle Verbindungen mit dieser Ressource.

fForce: Legt fest, ob geöffnete Dateien oder Druckaufträge auf dem Gerät vor der Unterbrechung der Anbindung geschlossen

werden. Wenn dieser Parameter *FALSE* ist und geöffnete Dateien oder Druckaufträge existieren, sollte die Verbindung nicht abgebrochen werden und die Funktion den Fehlerwert *WN_OPEN_FILES* zurückliefern.

Rückgabewert Der Rückgabewert kann einer der folgenden Werte sein:

Wert	Bedeutung
WN_SUCCESS	Die Funktion war erfolgreich.
WN_NOT_-SUPPORTED	Die Funktion wurde nicht unterstützt.
WN_OUT_OF_-MEMORY	Nicht genügend Speicherplatz im System.
WN_NET_ERROR	Ein Netzfehler ist aufgetreten.
WN_BAD_POINTER	Der Zeiger war ungültig.
WN_BAD_VALUE	Der Parameter *lpszName* war kein gültiger Name eines lokalen Geräts oder Netzes.
WN_NOT_-CONNECTED	Der Parameter *lpszName* enthielt kein umgeleitetes lokales Laufwerk oder aktuell benutzte Netzressource.
WN_OPEN_FILES	Die Anbindung wurde nicht unterbrochen. Der Parameter *fForce* hatte den Wert *FALSE* und es waren Dateien geöffnet.

Siehe auch *WNetAddConnection, WNetGetConnection*

WNetGetConnection 3.1

Syntax

```
UINT WNetGetConnection(lpszLocalName, lpszRemoteName,
   cbRemoteName)

LPSTR lpszLocalName;     /* Adresse des Namens des
                            lokalen Geräts          */
LPSTR lpszRemoteName     /* Adresse des Namens des
                            Remote-Geräts           */
UINT FAR* cbRemoteName;  /* Maximale Anzahl Byte
                            im Puffer               */
```

Beschreibung Die Funktion *WNetGetConnection* gibt den Namen der Netzressource zurück, die mit dem angegebenen umgeleiteten lokalen Gerät verbunden ist.

Parameter *lpszLocalName:* Zeigt auf einen nullterminierten String, der den Namen des umgeleiteten lokalen Geräts angibt.

lpszRemoteName: Zeigt auf den Puffer, der den nullterminierten Namen der Remote-Netzressource aufnimmt.

cbRemoteName: Zeigt auf eine Variable, die die maximale Anzahl Byte in dem durch *lpszRemoteName* bezeichneten Puffer bestimmt. Diese Funktion setzt die Variable auf die Anzahl Byte, die in diesen Puffer kopiert wurden.

Rückgabewert Der Rückgabewert kann einer der folgenden Werte sein:

Wert	Bedeutung
WN_SUCCESS	Die Funktion war erfolgreich.
WN_NOT_-SUPPORTED	Die Funktion wurde nicht unterstützt.
WN_OUT_OF_-MEMORY	Nicht genügend Speicherplatz im System.
WN_NET_ERROR	Ein Netzfehler ist aufgetreten.
WN_BAD_POINTER	Der Zeiger war ungültig.
WN_BAD_VALUE	Der Parameter *szLocalName* enthielt kein gültiges lokales Gerät.
WN_NOT_-CONNECTED	Der Parameter *szLocalName* war kein umgeleitetes lokales Gerät.
WN_MORE_DATA	Der Puffer war zu klein.

Siehe auch *WNetAddConnection, WNetCancelConnection*

WordBreakProc 3.1

Syntax

```
int CALLBACK WordBreakProc(lpszEditText,
ichCurrentWord, cbEditText, action)

LPSTR lpszEditText;      /* Adresse des Texts         */
int ichCurrentWord;      /* Index der Startposition   */
int cbEditText;          /* Länge des Texts           */
int action;              /* auszuführende Operation   */
```

Beschreibung Die Funktion *WordBreakProc* ist eine anwendungsdefinierte Callback-Funktion, die vom System immer dann aufgerufen wird, wenn eine Textzeile in einem mehrzeiligen Editierfeld umbrochen werden muß.

Parameter *lpszEditText:* Zeigt auf den Text im Editierfeld.

ichCurrentWord: Bestimmt einen Index auf ein Wort im Puffer des Texts, der den Punkt bezeichnet, an dem die Funktion die Überprüfung für einen Wortumbruch beginnen soll.

cbEditText: Bestimmt die Anzahl Byte im Text.

action: Bestimmt die durch die Callback-Funktion auszuführende Operation. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Operation
WB_LEFT	Den Anfang eines Worts links von der aktuellen Position ermitteln.
WB_RIGHT	Den Anfang eines Worts rechts von der aktuellen Position ermitteln.
WB_ISDELIMITER	Überprüfen, ob das Zeichen an der aktuellen Position ein Trennzeichen ist.

Rückgabewert

Wenn der Parameter *action* den Wert *WB_ISDELIMITER* hat, gibt die Funktion ungleich Null (*TRUE*) zurück, wenn das Zeichen an der aktuellen Position ein Trennzeichen ist. Andernfalls ist der Rückgabewert ein Index auf den Anfang eines Worts im Textpuffer.

Hinweis

Die Kombination von Wagenrücklauf (CR) und Zeilenvorschub (LF) muß von der Callback-Funktion als einzelnes Wort behandelt werden. Zwei Wagenrückläufe, gefolgt von einem Zeilenvorschub, müssen ebenfalls als einzelnes Wort behandelt werden.

Anwendungen müssen die Callback-Funktion installieren, indem sie die Adresse der Prozedurinstanz der Callback-Funktion in einer *EM_SETWORDBREAKPROC*-Botschaft angeben.

WordBreakProc ist ein Platzhalter für den bibliotheksdefinierten Funktionsnamen. Der tatsächliche Name muß in einer EXPORTS-Anweisung in der Moduldefinitionsdatei der Bibliothek exportiert werden.

Siehe auch

SendMessage

5. Datentypen

Die Datentypen in diesem Kapitel sind Schlüsselwörter. Sie definieren Größe und Bedeutung von Parametern und Rückgabewerten der verschiedenen Funktionen von Microsoft Windows 3.1. Die folgende Tabelle enthält Zeichen-, Integer-, Boolean- und Zeigertypen, sowie Handles. Die Zeichen-, Integer- und Boolean-Typen entsprechen denen der meisten C-Compiler. Die Namen der Zeigertypen besitzen häufig ein Präfix: *P* und *N* für *near*-Zeiger oder *LP* für *long*-Zeiger. Ein *near*-Zeiger greift auf Daten im aktuellen Datensegment zu. Ein *long*-Zeiger enthält einen 32-Bit-Wert Segment:Offset. Windows-Anwendungen verwenden Handles, um Ressourcen zu referenzieren, die in den Arbeitsspeicher geladen wurden. Windows ermöglicht den Zugriff auf diese Ressourcen über intern verwaltete Tabellen, die individuelle Einträge für jedes Handle enthalten. Jeder Eintrag in der Handle-Tabelle enthält die Adresse der Ressource und eine Bezeichnung des Ressourcentyps.

Die Windows-Datentypen sind in der folgenden Tabelle beschrieben:

Typ	Beschreibung
ABORTPROC	32-Bit-Zeiger auf eine *AbortProc*-Callback-Funktion.
ATOM	16-Bit-Wert, der als Atom-Handle benutzt wird.
BOOL	Boolescher 16-Bit-Wert.
BYTE	Vorzeichenloser 8-Bit-Integer. Verwenden Sie *LPBYTE*, um 32-Bit-Zeiger zu erzeugen. Verwenden Sie *PBYTE*, um Zeiger zu erzeugen, die dem verwendeten Speichermodell des Compilers entsprechen.
CATCHBUF[*9*]	18-Byte-Puffer, der von der Funktion *Catch* benutzt wird.
COLORREF	32-Bit-Wert, der als Farbwert benutzt wird.
DLGPROC	32-Bit-Zeiger auf eine Dialogfensterfunktion.
DWORD	Vorzeichenloser 32-Bit-Integer oder eine Segment:Offset-Adresse. Verwenden Sie *LPDWORD*, um 32-Bit-Zeiger zu erzeugen. Verwenden Sie *PDWORD*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
FARPROC	32-Bit-Zeiger auf eine Funktion.

Typ	Beschreibung
FNCALLBACK	32-Bit-Wert, der die *DdeCallback*-Funktion bezeichnet. Verwenden Sie *PFNCALLBACK*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
FONTENUMPROC	32-Bit-Zeiger auf eine *EnumFontsProc*-Callback-Funktion.
GLOBALHANDLE	16-Bit-Wert, der als Handle eines globalen Speicherobjekts benutzt wird.
GNOTIFYPROC	32-Bit-Zeiger auf eine *NotifyProc*-Callback-Funktion.
GOBJENUMPROC	32-Bit-Zeiger auf eine *EnumObjectsProc*-Callback-Funktion.
GRAYSTRINGPROC	32-Bit-Zeiger auf eine *GrayStringProc*-Callback-Funktion.
HANDLE	16-Bit-Wert, der als allgemeines Handle benutzt wird. Verwenden Sie *LPHANDLE*, um 32-Bit-Zeiger zu erzeugen. Mit *SPHANDLE* erzeugen Sie 16-Bit-Zeiger und mit *PHANDLE* Zeiger, die dem Speichermodell des Compilers entsprechen.
HCURSOR	16-Bit-Wert, der als Mauszeiger-Handle benutzt wird.
HFILE	16-Bit-Wert, der als Datei-Handle benutzt wird.
HGDIOBJ	16-Bit-Wert, der als Handle eines Objekts der GDI (grafische Geräteschnittstelle) benutzt wird.
HGLOBAL	16-Bit-Wert, der als Handle eines globalen Speicherobjekts benutzt wird.
HHOOK	32-Bit-Wert, der als Handle für Filter benutzt wird.
HKEY	32-Bit-Wert, der als Handle für ein Schlüsselwort in der Registrierungsdatenbank benutzt wird. Verwenden Sie *PHKEY*, um einen 32-Bit-Zeiger zu erzeugen.
HLOCAL	16-Bit-Wert, der als Handle für ein lokales Speicherobjekt benutzt wird.
HMODULE	16-Bit-Wert, der als Modul-Handle benutzt wird.
HOBJECT	16-Bit-Wert, der als Handle für ein OLE-Objekt benutzt wird.
HWND	16-Bit-Wert, der als Fenster-Handle benutzt wird.
HOOKPROC	32-Bit-Zeiger auf eine Filterfunktion.
HRSRC	16-Bit-Wert, der als Ressourcen-Handle benutzt wird.
LHCLIENTDOC	32-Bit-Wert, der als Handle für ein OLE-Client-Dokument benutzt wird.
LHSERVER	32-Bit-Wert, der als Handle für einen OLE-Server benutzt wird.

Typ	Beschreibung
LHSERVERDOC	32-Bit-Wert, der als Handle für ein OLE-Server-Dokument benutzt wird.
LINEDDAPROC	32-Bit-Zeiger auf eine *LineDDAProc*-Callback-Funktion.
LOCALHANDLE	16-Bit-Wert, der als Handle für ein lokales Speicherobjekt benutzt wird.
LONG	Vorzeichenbehafteter 32-Bit-Integer.
LPABC	32-Bit-Zeiger auf eine *ABC*-Struktur.
LPARAM	Vorzeichenbehafteter 32-Bit-Wert, der als Argument an eine Fensterfunktion oder Callback-Funktion übergeben wird.
LPBI	32-Bit-Zeiger auf eine *BANDINFOSTRUCT*-Struktur.
LPBITMAP	32-Bit-Zeiger auf eine *BITMAP*-Struktur. Verwenden Sie *NPBITMAP*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PBITMAP*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPBITMAPCOREHEADER	32-Bit-Zeiger auf eine *BITMAPCOREHEADER*-Struktur. Verwenden Sie *PBITMAPCOREHEADER*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPBITMAPCOREINFO	32-Bit-Zeiger auf eine *BITMAPCOREINFO*-Struktur. Verwenden Sie *PBITMAPCOREINFO*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPBITMAPFILEHEADER	32-Bit-Zeiger auf eine *BITMAPFILEHEADER*-Struktur. Verwenden Sie *PBITMAPFILEHEADER*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPBITMAPINFO	32-Bit-Zeiger auf eine *BITMAPINFO*-Struktur. Verwenden Sie *PBITMAPINFO*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPBITMAPINFOHEADER	32-Bit-Zeiger auf eine *BITMAPINFOHEADER*-Struktur. Verwenden Sie *PBITMAPINFOHEADER*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPCATCHBUF	32-Bit-Zeiger auf ein *CATCHBUF*-Array.
LPCBT_CREATEWND	32-Bit-Zeiger auf eine *CBT_CREATEWND*-Struktur.
LPCHOOSECOLOR	32-Bit-Zeiger auf eine *CHOOSECOLOR*-Struktur.
LPCHOOSEFONT	32-Bit-Zeiger auf eine *CHOOSEFONT*-Struktur.
LPCLIENTCREATESTRUCT	32-Bit-Zeiger auf eine *CLIENTCREATESTRUCT*-Struktur.

Typ	Beschreibung
LPCOMPAREITEMSTRUCT	32-Bit-Zeiger auf eine *COMPAREITEMSTRUCT*-Struktur. Verwenden Sie *PCOMPAREITEMSTRUCT*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPCPLINFO	32-Bit-Zeiger auf eine *CPLINFO*-Struktur. Verwenden Sie *PCPLINFO*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPCREATESTRUCT	32-Bit-Zeiger auf eine *CREATESTRUCT*-Struktur.
LPCSTR	32-Bit-Zeiger auf einen nichtmodifizierbaren Zeichen-String.
LPCTLINFO	32-Bit-Zeiger auf eine *CTLINFO*-Struktur. Verwenden Sie *PCTLINFO*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPCTLSTYLE	32-Bit-Zeiger auf eine *CTLSTYLE*-Struktur. Verwenden Sie *PCTLSTYLE*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPDCB	32-Bit-Zeiger auf eine *DCB*-Struktur.
LPDEBUGHOOKINFO	32-Bit-Zeiger auf eine *DEBUGHOOKINFO*-Struktur.
LPDELETEITEMSTRUCT	32-Bit-Zeiger auf eine *DELETEITEMSTRUCT*-Struktur. Verwenden Sie *PDELETEITEMSTRUCT*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPDEVMODE	32-Bit-Zeiger auf eine *DEVMODE*-Struktur. Verwenden Sie *NPDEVMODE*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PDEVMODE*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPDEVNAMES	32-Bit-Zeiger auf eine *DEVNAMES*-Struktur.
LPDOCINFO	32-Bit-Zeiger auf eine *DOCINFO*-Struktur.
LPDRAWITEMSTRUCT	32-Bit-Zeiger auf eine *DRAWITEMSTRUCT*-Struktur. Verwenden Sie *PDRAWITEMSTRUCT*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPDRIVERINFOSTRUCT	32-Bit-Zeiger auf eine *DRIVERINFOSTRUCT*-Struktur.
LPDRVCONFIGINFO	32-Bit-Zeiger auf eine *DRVCONFIGINFO*-Struktur. Verwenden Sie *PDRVCONFIGINFO*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPEVENTMSG	32-Bit-Zeiger auf eine *EVENTMSG*-Struktur. Verwenden Sie *NPEVENTMSG*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PEVENTMSG*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPDRIVERINFOSTRUCT	32-Bit-Zeiger auf eine *DRIVERINFOSTRUCT*-Struktur.

Typ	Beschreibung
LPFINDREPLACE	32-Bit-Zeiger auf eine *FINDREPLACE*-Struktur.
LPFMS_GETDRIVEINFO	32-Bit-Zeiger auf eine *FMS_GETDRIVEINFO*-Struktur.
LPFMS_GETFILESEL	32-Bit-Zeiger auf eine *FMS_GETFILESEL*-Struktur.
LPFMS_LOAD	32-Bit-Zeiger auf eine *FMS_LOAD*-Struktur.
LPHANDLETABLE	32-Bit-Zeiger auf eine *HANDLETABLE*-Strukur. Verwenden Sie *PHANDLETABLE*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPHELPWININFO	32-Bit-Zeiger auf eine *HELPWININFO*-Struktur. Verwenden Sie *PHELPWININFO*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPINT	32-Bit-Zeiger auf einen vorzeichenbehafteten 16-Bit-Wert. Verwenden Sie *PINT*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPKERNINGPAIR	32-Bit-Zeiger auf eine *KERNINGPAIR*-Struktur.
LPLOGBRUSH	32-Bit-Zeiger auf eine *LOGBRUSH*-Struktur. Verwenden Sie *NPLOGBRUSH*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PLOGBRUSH*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPLOGFONT	32-Bit-Zeiger auf eine *LOGFONT*-Struktur. Verwenden Sie *NPLOGFONT*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PLOGFONT*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPLOGPALETTE	32-Bit-Zeiger auf eine *LOGPALETTE*-Struktur. Verwenden Sie *NPLOGPALETTE*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PLOGPALETTE*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPLOGPEN	32-Bit-Zeiger auf eine *LOGPEN*-Struktur. Verwenden Sie *NPLOGPEN*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PLOGPEN*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPLONG	32-Bit-Zeiger auf einen vorzeichenbehafteten 32-Bit-Integer. Verwenden Sie *PLONG*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPMAT2	32-Bit-Zeiger auf eine *MAT2*-Struktur.
LPMDICREATESTRUCT	32-Bit-Zeiger auf eine *MDICREATESTRUCT*-Struktur.
LPMEASUREITEMSTRUCT	32-Bit-Zeiger auf eine *MEASUREITEMSTRUCT*-Struktur. Verwenden Sie *PMEASUREITEMSTRUCT*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.

Typ	Beschreibung
LPMETAFILEPICT	32-Bit-Zeiger auf eine *METAFILEPICT*-Struktur.
LPMETARECORD	32-Bit-Zeiger auf eine *METARECORD*-Struktur. Verwenden Sie *PMETARECORD*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPMOUSEHOOKSTRUCT	32-Bit-Zeiger auf eine *MOUSEHOOKSTRUCT*-Struktur.
LPMSG	32-Bit-Zeiger auf eine *MSG*-Struktur. Verwenden Sie *NPMSG*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PMSG*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPNCCALCSIZE_PARAMS	32-Bit-Zeiger auf eine *NCCALCSIZE_PARAMS*-Struktur.
LPNEWCPLINFO	32-Bit-Zeiger auf eine *NEWCPLINFO*-Struktur. Verwenden Sie *PNEWCPLINFO*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPNEWTEXTMETRIC	32-Bit-Zeiger auf eine *NEWTEXTMETRIC*-Struktur. Verwenden Sie *NPNEWTEXTMETRIC*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PNEWTEXTMETRIC*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPOFSTRUCT	32-Bit-Zeiger auf eine *OFSTRUCT*-Struktur. Verwenden Sie *NPOFSTRUCT*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *POFSTRUCT*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPOLECLIENT	32-Bit-Zeiger auf eine *OLECLIENT*-Struktur.
LPOLECLIENTVTBL	32-Bit-Zeiger auf eine *OLECLIENTVTBL*-Struktur.
LPOLEOBJECT	32-Bit-Zeiger auf eine *OLEOBJECT*-Struktur.
LPOLEOBJECTVTBL	32-Bit-Zeiger auf eine *OLEOBJECTVTBL*-Struktur.
LPOLESERVER	32-Bit-Zeiger auf eine *OLESERVER*-Struktur.
LPOLESERVERDOC	32-Bit-Zeiger auf eine *OLESERVERDOC*-Struktur.
LPOLESERVERDOCVTBL	32-Bit-Zeiger auf eine *OLESERVERDOCVTBL*-Struktur.
LPOLESERVERVTBL	32-Bit-Zeiger auf eine *OLESERVERVTBL*-Struktur.
LPOLESTREAM	32-Bit-Zeiger auf eine *OLESTREAM*-Struktur.
LPOLESTREAMVTBL	32-Bit-Zeiger auf eine *OLESTREAMVTBL*-Struktur.
LPOLETARGETDEVICE	32-Bit-Zeiger auf eine *OLETARGETDEVICE*-Struktur.
LPOPENFILENAME	32-Bit-Zeiger auf eine *OPENFILENAME*-Struktur.
LPOUTLINETEXTMETRIC	32-Bit-Zeiger auf eine *OUTLINETEXTMETRIC*-Struktur.

Typ	Beschreibung
LPPAINTSTRUCT	32-Bit-Zeiger auf eine *PAINTSTRUCT*-Struktur. Verwenden Sie *NPPAINTSTRUCT*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PPAINTSTRUCT*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPPALETTEENTRY	32-Bit-Zeiger auf eine *PALETTEENTRY*-Struktur.
LPPOINT	32-Bit-Zeiger auf eine *POINT*-Struktur. Verwenden Sie *NPPOINT*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PPOINT*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPPOINTFX	32-Bit-Zeiger auf eine *POINTFX*-Struktur.
LPPRINTDLG	32-Bit-Zeiger auf eine *PRINTDLG*-Struktur.
LPRASTERIZER_STATUS	32-Bit-Zeiger auf eine *RASTERIZER_STATUS*-Struktur.
LPRECT	32-Bit-Zeiger auf eine *RECT*-Struktur. Verwenden Sie *NPRECT*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PRECT*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPRGBQUAD	32-Bit-Zeiger auf eine *RGBQUAD*-Struktur.
LPRGBTRIPLE	32-Bit-Zeiger auf eine *RGBTRIPLE*-Struktur.
LPSEGINFO	32-Bit-Zeiger auf eine *SEGINFO*-Struktur.
LPSIZE	32-Bit-Zeiger auf eine *SIZE*-Struktur. Verwenden Sie *NPSIZE*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PSIZE*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPSTR	32-Bit-Zeiger auf eine *STR*-Struktur. Verwenden Sie *NPSTR*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PSTR*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPTEXTMETRIC	32-Bit-Zeiger auf eine *TEXTMETRIC*-Struktur. Verwenden Sie *NPTEXTMETRIC*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PTEXTMETRIC*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPTTPOLYCURVE	32-Bit-Zeiger auf eine *TTPOLYCURVE*-Struktur.
LPTTPOLYGONHEADER	32-Bit-Zeiger auf eine *TTPOLYGONHEADER*-Struktur.
LPVOID	32-Bit-Zeiger auf einen unbestimmten Typ.
LPWINDOWPLACEMENT	32-Bit-Zeiger auf eine *WINDOWPLACEMENT*-Struktur. Verwenden Sie *PWINDOWPLACEMENT*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.

Typ	Beschreibung
LPWINDOWPOS	32-Bit-Zeiger auf eine *WINDOWPOS*-Struktur.
LPWNDCLASS	32-Bit-Zeiger auf eine *WNDCLASS*-Struktur. Verwenden Sie *NPWNDCLASS*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PWNDCLASS*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LPWORD	32-Bit-Zeiger auf einen vorzeichenlosen 16-Bit-Wert. Verwenden Sie *PWORD*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
LRESULT	Vorzeichenbehafteter 32-Bit-Wert, der von einer Fensterfunktion oder Callback-Funktion zurückgegeben wird.
MFENUMPROC	32-Bit-Zeiger auf eine *EnumMetaFileProc*-Callback-Funktion.
NEARPROC	16-Bit-Zeiger auf eine Funktion.
OLECLIPFORMAT	16-Bit-Wert, der als Standardformat der Zwischenablage benutzt wird.
PATTERN	Äquivalent zur *LOGBRUSH*-Struktur. Verwenden Sie *LPPATTERN*, um 32-Bit-Zeiger und *NPATTERN*, um 16-Bit-Zeiger zu erzeugen. Verwenden Sie *PPATTERN*, um Zeiger zu erzeugen, die dem Speichermodell des Compilers entsprechen.
PCONVCONTEXT	32-Bit-Zeiger auf eine *CONVCONTEXT*-Struktur.
PCONVINFO	32-Bit-Zeiger auf eine *CONVINFO*-Struktur.
PHSZPAIR	32-Bit-Zeiger auf eine *HSZPAIR*-Struktur.
PROPENUMPROC	32-Bit-Zeiger auf eine *EnumPropFixedProc*- oder *EnumPropMoveableProc*-Callback-Funktion.
RSRCHDLRPROC	32-Bit-Zeiger auf eine *LoadProc*-Callback-Funktion.
TIMERPROC	32-Bit-Zeiger auf eine *TimerProc*-Callback-Funktion.
UINT	Vorzeichenloser 16-Bit-Wert.
WNDENUMPROC	32-Bit-Zeiger auf eine *EnumWindowsProc*-Callback-Funktion.
WNDPROC	32-Bit-Zeiger auf eine Fensterfunktion.
WORD	Vorzeichenloser 16-Bit-Wert.
WPARAM	Vorzeichenbehafteter 16-Bit-Wert, der als Argument an eine Fensterfunktion oder Callback-Funktion übergeben wird.

6. Botschaften

CB_ADDSTRING 3.0

Syntax

```
CB_ADDSTRING
wParam = 0;                          /* nicht verwendet,
                                        muß Null sein     */
lParam = (LPARAM) (LPCSTR) lpsz; /* Adresse des hinzuzu-
                                        fügenden String   */
```

Beschreibung

Anwendungen übergeben eine *CB_ADDSTRING*-Botschaft, um der Liste eines Kombinationsfensters einen String hinzuzufügen. Wenn die Liste nicht den Stil *CBS_SORT* hat, wird der String am Ende der Liste eingefügt. Andernfalls wird der String in die Liste eingefügt und die Liste neu sortiert.

Parameter

lpsz: Wert von *lParam*. Zeigt auf den nullterminierten String, der hinzugefügt werden soll. Wurde das Kombinationsfenster mit einem besitzergezeichneten Stil aber ohne *CBS_HASSTRINGS* erstellt, wird der Wert des Parameters *lpsz* gespeichert und nicht der String, auf den er andernfalls zeigen würde.

Rückgabewert

Der Rückgabewert ist der von Null ausgehende Index des String in der Liste. Der Rückgabewert ist *CB_ERR*, wenn ein Fehler auftritt. Er ist *CB_ERRSPACE*, wenn der Speicherplatz nicht ausreicht, um den neuen String zu speichern.

Hinweis

Wenn ein besitzergezeichnetes Kombinationsfenster mit dem Stil *CBS_SORT* aber ohne *CBS_HASSTRINGS* erzeugt wurde, wird die Botschaft *WM_COMPAREITEM* ein- oder mehrmals an den Besitzer des Kombinationsfensters übergeben, damit das neue Element richtig in die Liste eingeordnet werden kann.

Um einen String an einer bestimmten Position in eine Liste einzufügen, kann die Botschaft *CB_INSERTSTRING* verwendet werden.

Beispiel

Dieses Beispiel fügt den String »Mein String« in eine Liste ein:

```
DWORD dwIndex;
dwIndex = SendDlgItemMessage(hdlg, ID_MYCOMBOBOX,
   CB_ADDSTRING, 0, (LPARAM) ((LPCSTR) "Mein String"));
```

Siehe auch *CB_INSERTSTRING, WM_COMPAREITEM*

CB_DELETESTRING 3.0

Syntax

```
CB_DELETESTRING

wParam = (WPARAM) index;  /* zu löschendes Element    */
lParam = 0L;              /* nicht verwendet,
                             muß Null sein            */
```

Beschreibung Anwendungen übergeben eine *CB_DELETESTRING*-Botschaft, um einen String aus der Liste eines Kombinationsfensters zu löschen.

Parameter *index:* Wert von *wParam*. Bestimmt den von Null ausgehenden Index des zu löschenden String.

Rückgabewert Der Rückgabewert ist die Anzahl der in der Liste verbleibenden Strings. Der Rückgabewert ist *CB_ERR*, wenn der Parameter *index* einen Index angibt, der größer als die Anzahl der Elemente in der Liste ist.

Hinweis Wenn das Kombinationsfenster mit einem besitzergezeichneten Stil, aber ohne *CBS_HASSTRINGS* erzeugt wurde, wird eine *WM_DELETEITEM*-Botschaft an den Besitzer des Kombinationsfensters übergeben, damit die Anwendung alle mit dem Element verknüpften Daten freigeben kann.

Beispiel Dieses Beispiel löscht den ersten String in einem Kombinationsfenster:

```
DWORD dwRemaining;

dwRemaining = SendDlgItemMessage(hdlg, ID_MYCOMBOBOX,
   CB_DELETESTRING, 0, 0L);
```

Siehe auch *WM_DELETEITEM*

CB_FINDSTRINGEXACT 3.1

Syntax

```
CB_FINDSTRINGEXACT

wParam = (WPARAM) indexStart;    /* Element vor dem
                                    Beginn der Suche  */
lParam = (LPARAM) (LPCSTR) lpszFind;
                                 /* Adresse des
                                    Präfix-String     */
```

Beschreibung Anwendungen übergeben eine *CB_FINDSTRINGEXACT*-Botschaft, um den ersten String in der Liste eines Kombinationsfensters zu finden, der dem im Parameter *lpszFind* angegebenen String entspricht.

Parameter *indexStart:* Wert von *wParam*. Bestimmt den von Null ausgehenden Index des Elements vor dem ersten zu untersuchenden Element. Wenn die Suche das Ende der Liste erreicht, beginnt sie am Anfang der Liste bis zu dem Element, das durch *indexStart* angegeben ist. Wenn *indexStart* -1 ist, wird die gesamte Liste von Anfang an durchsucht.

lpszFind: Wert von *lParam*. Zeigt auf den nullterminierten String, nach dem gesucht werden soll. Dieser String kann einen vollständigen Dateinamen, einschließlich der Namenserweiterung, enthalten. Die Suche berücksichtigt Klein- und Großschreibung nicht. Der String kann also jede Kombination von Klein- und Großbuchstaben enthalten.

Rückgabewert Der Rückgabewert ist der von Null ausgehende Index des gefundenen Elements. Er ist *CB_ERR*, wenn die Suche nicht erfolgreich war.

Hinweis Wenn das Kombinationsfenster mit einem besitzergezeichneten Stil, aber ohne *CBS_HASSTRINGS* erzeugt wurde, gibt diese Botschaft den Index des Elements zurück, dessen *DWORD*-Wert dem Wert des Parameters *lpszFind* entspricht.

Siehe auch *CB_FINDSTRING, CB_SETCURSEL*

CB_GETDROPPEDCONTROLRECT 3.1

Syntax

```
CB_GETDROPPEDCONTROLRECT

wParam = 0;                          /* nicht verwendet,
                                        muß Null sein  */
lParam = (LPARAM) (RECT FAR*) lprc; /* Adresse der RECT-
                                        Struktur       */
```

Beschreibung Anwendungen übergeben eine *CB_GETDROPPEDCONTROLRECT*-Botschaft, um die Bildschirmkoordinaten der sichtbaren Liste eines Kombinationsfensters abzurufen.

Parameter *lprc:* Wert von *lParam*. Zeigt auf die *RECT*-Struktur, die die Koordinaten aufnehmen soll. Die *RECT*-Struktur hat folgende Form:

```
typedef struct tagRECT { /* rc */
   int left;
   int top;
   int right;
   int bottom;
} RECT;
```

Rückgabewert Der Rückgabewert ist immer *CB_OKAY*.

Beispiel Dieses Beispiel ruft das umgebende Rechteck der Liste eines Kombinationsfenster ab:

```
RECT rcl;

SendDlgItemMessage(hdlg, ID_MYCOMBOBOX,
   CB_GETDROPPEDCONTROLRECT, 0, (DWORD) ((LPRECT)
&rcl));
```

CB_GETDROPPEDSTATE 3.1

Syntax

```
CB_GETDROPPEDSTATE

wParam = 0;       /* nicht verwendet, muß Null sein   */
lParam = 0L;      /* nicht verwendet, muß Null sein   */
```

Beschreibung Anwendungen übergeben eine *CB_GETDROPPEDSTATE*-Botschaft, um zu ermitteln, ob die Liste eines Kombinationsfensters sichtbar ist.

Parameter Diese Botschaft verwendet keine Parameter.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die Liste sichtbar ist, andernfalls ist er Null.

Beispiel Dieses Beispiel ermittelt, ob die Liste eines Kombinationsfensters sichtbar ist:

```
BOOL fDropped;

fDropped = (BOOL) SendDlgItemMessage(hdlg,
   ID_MYCOMBOBOX, CB_GETDROPPEDSTATE, 0, 0L);
```

Siehe auch *CB_SHOWDROPDOWN*

CB_GETEXTENDEDUI 3.1

Syntax

```
CB_GETEXTENDEDUI

wParam = 0;       /* nicht verwendet, muß Null sein   */
lParam = 0L;      /* nicht verwendet, muß Null sein   */
```

Beschreibung Anwendungen übergeben eine *CB_GETEXTENDEDUI*-Botschaft, um zu ermitteln, ob ein Kombinationsfenster die Standard- oder die erweiterte Benutzerschnittstelle hat.

Parameter Diese Botschaft verwendet keine Parameter.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn das Kombinationsfenster die erweiterte Benutzerschnittstelle hat. Andernfalls ist er Null.

Hinweis Die erweiterte Benutzerschnittstelle unterscheidet sich von der Standardschnittstelle in folgenden Punkten:

- Das Anklicken eines statischen Steuerelements stellt die Liste dar (nur für den Stil *CBS_DROPDOWNLIST*).

- Das Drücken der Taste ⊟ stellt die Liste dar (F4 ist deaktiviert).
- Der Bildlauf im statischen Steuerelement ist deaktiviert, wenn die Liste nicht sichtbar ist (die Pfeiltasten sind deaktiviert).

Beispiel

Dieses Beispiel ermittelt, ob ein Kombinationsfenster die erweiterte Benutzerschnittstelle hat:

```
BOOL fExtended;

fExtended = (BOOL) SendDlgItemMessage(hdlg,
   ID_MYCOMBOBOX, CB_GETEXTENDEDUI, 0, 0L);
```

Siehe auch

CB_SETEXTENDEDUI

CB_GETITEMHEIGHT 3.1

Syntax

```
CB_GETITEMHEIGHT

wParam = (WPARAM) index;   /* Index des Elements      */
lParam = 0L;               /* nicht verwendet, muß
                              Null sein               */
```

Beschreibung

Anwendungen übergeben die Botschaft *CB_GETITEMHEIGHT*, um die Höhe der Elemente in der Liste eines Kombinationsfensters zu ermitteln.

Parameter

index: Wert von *wParam*. Bestimmt die Komponente des Kombinationsfensters, deren Höhe abgerufen werden soll. Ist *index* -1, wird die Höhe des statischen Textfelds oder des Editierfelds des Kombinationsfensters zurückgegeben. Hat das Kombinationsfenster den Stil *CBS_OWNERDRAWVARIABLE*, bezeichnet *index* den von Null ausgehenden Index des Elements, dessen Höhe zurückgeliefert werden soll. In allen anderen Fällen sollte *index* auf Null gesetzt werden.

Rückgabewert

Der Rückgabewert ist die Höhe der Elemente in der Liste eines Kombinationsfensters in Pixel. Hat das Kombinationsfenster den Stil *CBS_OWNERDRAWVARIABLE*, ist der Rückgabewert die Höhe des durch *index* bestimmten Elements. Ist *index* -1, entspricht der Rückgabewert der Höhe des Editierfelds bzw. des statischen Textfelds im Kombinationsfenster. Der Rückgabewert ist *CB_ERR*, wenn ein Fehler auftritt.

Beispiel

Dieses Beispiel sendet eine *CB_GETITEMHEIGHT*-Botschaft, um die Höhe der Elemente in der Liste eines Kombinationsfensters abzurufen:

```
LRESULT lrHeight;
lrHeight = SendDlgItemMessage(hdlg, ID_MYCOMBOBOX,
   CB_GETITEMHEIGHT, 0, 0L);
```

Siehe auch *CB_SETITEMHEIGHT*

CB_SETEXTENDEDUI 3.1

Syntax

```
CB_SETEXTENDEDUI

wParam = (WPARAM) (BOOL) fExtended;
                  /* Flag für die erweiterte
                     Benutzerschnittstelle         */
lParam = 0L;      /* nicht verwendet, muß Null sein */
```

Beschreibung Anwendungen übergeben die Botschaft *CB_SETEXTENDEDUI*, um die Standard- oder die erweiterte Benutzerschnittstelle für ein Kombinationsfenster auszuwählen, das den Stil *CBS_DROPDOWN* oder *CBS_DROPDOWNLIST* hat.

Parameter *fExtended:* Wert von *wParam*. Legt fest, ob das Kombinationsfenster die Standard- oder die erweiterte Benutzerschnittstelle hat. Der Wert *TRUE* wählt die erweiterte, der Wert *FALSE* die Standardschnittstelle aus.

Rückgabewert Der Rückgabewert ist *CB_OK*, wenn die Aktion erfolgreich durchgeführt wurde. Andernfalls ist der Rückgabewert *CB_ERR*.

Hinweis Die erweiterte Benutzerschnittstelle unterscheidet sich von der Standardschnittstelle in folgenden Punkten:

- Das Anklicken eines statischen Steuerelements stellt die Liste dar (nur für den Stil *CBS_DROPDOWNLIST*).
- Das Drücken der Taste − stellt die Liste dar (F4 ist deaktiviert).
- Der Bildlauf im statischen Steuerelement ist deaktiviert, wenn die Liste nicht sichtbar ist (die Pfeiltasten sind deaktivert).

Beispiel Dieses Beispiel wählt die erweiterte Benutzerschnittstelle für ein Kombinationsfenster:

```
SendDlgItemMessage(hdlg, ID_MYCOMBOBOX,
   CB_SETEXTENDEDUI, TRUE, 0L);
```

Siehe auch *CB_GETEXTENDEDUI*

CB_SETITEMHEIGHT 3.1

Syntax

```
CB_SETITEMHEIGHT

wParam = (WPARAM) index;        /* Index des Elements */
lParam = (LPARAM) (int) height; /* Höhe des Elements  */
```

Beschreibung Anwendungen übergeben eine *CB_SETITEMHEIGHT*-Botschaft, um die Höhe der Elemente in der Liste eines Kombinationsfensters oder die Höhe des Editierfelds eines Kombinationsfensters zu setzen.

Parameter *index:* Wert von *wParam*. Bestimmt, ob die Höhe der Elemente in der Liste oder die Höhe des Editierfelds eines Kombinationsfensters gesetzt wird.

Hat das Kombinationsfenster den Stil *CBS_OWNERDRAWVARIABLE*, bestimmt der Parameter *index* den von Null ausgehenden Index des Elements, dessen Höhe gesetzt werden soll. Andernfalls muß *index* Null sein. Dann wird die Höhe aller Elemente gesetzt. Ist *index* -1, wird die Höhe des Editierfelds oder des statischen Texts eines Kombinationsfensters gesetzt.

height: Wert des niederwertigen *WORD* von *lParam*. Bestimmt die Höhe der durch *index* bezeichneten Kombinationsfensterkomponente in Pixel.

Rückgabewert Der Rückgabewert ist *CB_ERR*, wenn der Index oder die Höhe ungültig ist.

Hinweis Die Höhe des Editierfelds des Kombinationsfensters wird unabhängig von der Höhe der Elemente in der Liste gesetzt. Anwendungen müssen sicherstellen, daß die Höhe des Editierfelds nicht kleiner als die Höhe eines Elements ist.

Beispiel Dieses Beispiel übergibt eine *CB_SETITEMHEIGHT*-Botschaft, um die Höhe der Elemente in der Liste in einem Kombinationsfenster zu setzen:

```
LPARAM lrHeight;

SendDlgItemMessage(hdlg, ID_MYCOMBOBOX,
   CB_SETITEMHEIGHT, 0, lrHeight);
```

Siehe auch *CB_GETITEMHEIGHT*

EM_GETFIRSTVISIBLELINE 3.1

Syntax

```
EM_GETFIRSTVISIBLELINE

wParam = 0;         /* nicht verwendet, muß Null sein */
lParam = 0L;        /* nicht verwendet, muß Null sein */
```

Beschreibung Anwendungen übergeben eine *EM_GETFIRSTVISIBLELINE*-Botschaft, um die oberste sichtbare Zeile in einem Editierfeld zu ermitteln.

Parameter Diese Botschaft verwendet keine Parameter.

Rückgabewert Der Rückgabewert ist der von Null ausgehende Index der obersten sichtbaren Zeile. Für einzeilige Editierfelder wird Null zurückgegeben.

Beispiel Dieses Beispiel ermittelt den Index der obersten sichtbaren Zeile in einem Editierfeld:

```
int FirstVis;

FirstVis = (int) SendDlgItemMessage(hdlg, IDD_EDIT,
   EM_GETFIRSTVISIBLELINE, 0, 0L);
```

EM_GETPASSWORDCHAR 3.1

Syntax

```
EM_GETPASSWORDCHAR

wParam = 0;          /* nicht verwendet, muß Null sein */
lParam = 0L,         /* nicht verwendet, muß Null sein */
```

Beschreibung Anwendungen übergeben eine *EM_GETPASSWORDCHAR*-Botschaft, um das Zeichen abzurufen, das in Editierfeldern dargestellt wird, wenn der Anwender ein Paßwort eingibt.

Parameter Diese Botschaft verwendet keine Parameter.

Rückgabewert Der Rückgabewert bestimmt das Zeichen, das anstelle des eingegebenen Zeichens dargestellt wird. Der Rückgabewert ist *NULL*, wenn kein entsprechendes Zeichen existiert.

Wenn das Editierfeld mit dem Stil *ES_PASSWORD* erzeugt wurde, ist das voreingestellte Zeichen der Stern (*).

Siehe auch *EM_SETPASSWORDCHAR*

EM_GETWORDBREAKPROC 3.1

Syntax

```
EM_GETWORDBREAKPROC

wParam = 0;          /* nicht verwendet, muß Null sein */
lParam = 0L;         /* nicht verwendet, muß Null sein */
```

Beschreibung Anwendungen übergeben eine *EM_GETWORDBREAKPROC*-Botschaft an Editierfelder, um die aktuelle Wortumbruchfunktion abzurufen.

Parameter Diese Botschaft verwendet keine Parameter.

Rückgabewert Der Rückgabewert bestimmt die Adresse der Prozedurinstanz der anwendungsdefinierten Wortumbruchfunktion. Der Rückgabewert ist *NULL*, wenn keine Wortumbruchfunktion existiert.

Hinweis Eine Wortumbruchfunktion durchsucht einen Textpuffer nach dem ersten Wort, das nicht mehr in die aktuelle Zeile der Anzeige paßt. Die Wortumbruchfunktion positioniert dieses Wort auf dem Anzeigegerät am Anfang der folgenden Zeile. Wortumbruchfunktionen definieren den Punkt, an dem Windows Textzeilen in mehrzeiligen Editierfeldern umbricht. Dies

geschieht normalerweise an einem Leerzeichen, das zwei Wörter voneinander trennt.

Siehe auch *EM_SETWORDBREAKPROC, MakeProcInstance, WordBreakProc*

EM_SETREADONLY 3.1

Syntax

```
EM_SETREADONLY

wParam = (WPARAM) (BOOL) fReadOnly;
                              /* Nur-Lesen-Flag         */
lParam = 0L;                  /* nicht verwendet,
                                 muß Null sein         */
```

Beschreibung Anwendungen übergeben eine *EM_SETREADONLY*-Botschaft, um den Nur-Lesen-Status eines Editierfelds zu setzen.

Parameter *fReadOnly:* Wert von *wParam*. Bestimmt, ob der Nur-Lesen-Status eines Editierfelds gesetzt oder entfernt werden soll. Der Wert *TRUE* setzt den Status Nur-Lesen, der Wert *FALSE* setzt den Status Lesen/Schreiben.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die Funktion erfolgreich ist. Tritt ein Fehler auf, ist er Null.

Hinweis Wenn der Status eines Editerfelds auf Nur-Lesen gesetzt wird, kann der Anwender den Text innerhalb dieses Felds nicht verändern.

Beispiel Dieses Beispiel setzt den Status eines Editierfeldes auf Nur-Lesen:

```
SendDlgItemMessage(hdlg, IDD_EDIT, EM_SETREADONLY,
   TRUE, 0L);
```

EM_SETWORDBREAKPROC 3.1

Syntax

```
EM_SETWORDBREAKPROC

wParam = 0;          /* nicht verwendet, muß Null sein */
lParam = (LPARAM) (EDITWORDBREAKPROC) ewbprc;
                     /* Adresse der Funktion           */
```

Beschreibung Anwendungen übergeben eine *EM_SETWORDBREAKPROC*-Botschaft an Editierfelder, um die voreingestellte Wortumbruchfunktion durch eine anwendungsdefinierte Wortumbruchfunktion zu ersetzen.

Parameter *ewbprc:* Wert von *lParam*. Bestimmt die Adresse der Prozedurinstanz der anwendungsdefinierten Wortumbruchfunktion. Diese Adresse muß mit der Funktion *MakeProcInstance* erzeugt werden. Weitere Informationen finden Sie in der Beschreibung der Callback-Funktion *WordBreakProc*.

Rückgabewert Diese Botschaft hat keinen Rückgabewert.

Hinweis Eine Wortumbruchfunktion durchsucht einen Textpuffer nach dem ersten Wort, das nicht mehr in die aktuelle Zeile des Anzeigegeräts paßt. Die Wortumbruchfunktion positioniert dieses Wort am Anfang der nächsten Zeile.

Wortumbruchfunktionen definieren den Punkt, an dem Windows eine Zeile in mehrzeiligen Editierfeldern umbrechen soll. Dies ist normalerweise ein Leerzeichen, das zwei Wörter voneinander trennt. Die Funktion kann von mehrzeiligen oder einzeiligen Editierfeldern aufgerufen werden, wenn der Anwender die Pfeiltasten zusammen mit der Taste Strg drückt, um den Cursor auf das nächste Wort oder das vorhergehende Wort zu setzen.

Die voreingestellte Wortumbruchfunktion bricht eine Textzeile an einem Leerzeichen um. Die anwendungsdefinierte Funktion kann den Wortumbruch auch für Bindestriche oder andere Zeichen definieren.

Siehe auch *EM_GETWORDBREAKPROC, MakeProcInstance, WordBreakProc*

LB_FINDSTRINGEXACT 3.1

Syntax

```
LB_FINDSTRINGEXACT

wParam = (WPARAM) indexStart;        /* Element, hinter
                                        dem die Suche
                                        beginnen soll */
lParam = (LPARAM) (LPCSTR) lpszFind; /* Adresse des
                                        Such-String   */
```

Beschreibung Anwendungen übergeben eine *LB_FINDSTRINGEXACT*-Botschaft, um den ersten String in einer Liste zu finden, der dem im Parameter *lpszFind* angegebenen String entspricht.

Parameter *indexStart:* Wert von *wParam*. Bestimmt den von Null ausgehenden Index des Elements vor dem ersten Element, das untersucht werden soll. Wenn die Suche das Ende der Liste erreicht, beginnt sie am Anfang der Liste, bis das im Parameter *indexStart* angegebene Element erreicht ist. Wenn *indexStart* -1 ist, wird die ganze Liste von Anfang an durchsucht.

lpszFind: Wert von *lParam*. Zeigt auf den nullterminierten String, nach dem gesucht werden soll. Dieser String kann einen vollständigen Dateinamen einschließlich der Namenserweiterung enthalten. Die Suche berücksichtigt Klein- und Großschreibung nicht. Der String kann also jede Kombination von Klein- und Großbuchstaben enthalten.

Rückgabewert Der Rückgabewert ist der Index des gefundenen Elements oder *LB_ERR*, wenn die Suche nicht erfolgreich war.

Hinweis Wenn die Liste mit einem besitzergezeichneten Stil, aber ohne *LB_HASSTRINGS* erzeugt wurde, gibt diese Botschaft den Index des Elements zurück, dessen *DWORD*-Wert dem im Parameter *lpszFind* angegebenen Wert entspricht. Der *DWORD*-Wert wird im Parameter *lParam* der Botschaft *LB_ADDSTRING* oder *LB_INSERTSTRING* angegeben.

Siehe auch *LB_ADDSTRING, LB_FINDSTRING, LB_INSERTSTRING*

LB_GETCARETINDEX 3.1

Syntax

```
LB_GETCARETINDEX

wParam = 0;         /* nicht verwendet, muß Null sein */
lParam = 0L;        /* nicht verwendet, muß Null sein */
```

Beschreibung Anwendungen übergeben eine *LB_GETCARETINDEX*-Botschaft, um den Index des Elements mit dem fokussierenden Rechteck in einer Liste für Mehrfachauswahl zu ermitteln. Das Element kann, muß aber nicht ausgewählt sein.

Parameter Diese Botschaft verwendet keine Parameter.

Rückgabewert Der Rückgabewert ist der von Null ausgehende Index des Elements mit dem fokussierenden Rechteck. Handelt es sich um eine Liste für einfache Auswahl, ist der Rückgabewert der Index des ausgewählten Elements.

Beispiel Dieses Beispiel übergibt eine *LB_GETCARETINDEX*-Botschaft, um den Index des Elements mit dem fokussierenden Rechteck in der Liste abzurufen:

```
LRESULT lrIndex;

lrIndex = SendDlgItemMessage(hdlg, ID_MYLISTBOX,
   LB_GETCARETINDEX, 0, 0L);
```

Siehe auch *LB_SETCARETINDEX*

LB_SETCARETINDEX 3.1

Syntax

```
LB_SETCARETINDEX

wParam = (WPARAM) index;          /* Index des
                                     Elements          */
lParam = MAKELPARAM(fScroll, 0); /* Flag für den Bild-
                                     lauf des Elements */
```

Beschreibung Anwendungen übergeben eine *LB_SETCARETINDEX*-Botschaft, um das fokussierende Rechteck in einer Liste für Mehrfachauswahl auf das Element mit dem angegebenen Index zu setzen. Wenn das Element nicht sichtbar ist, wird es in den sichtbaren Bereich verschoben.

Parameter *index:* Wert von *wParam*. Bestimmt den von Null ausgehenden Index des Elements in der Liste, das das fokussierende Rechteck erhalten soll.

fScroll: Wert von *lParam*. Wenn dieser Wert Null ist, wird das Element vollständig in den sichtbaren Bereich verschoben. Ist dieser Wert ungleich Null, wird das Element nur solange verschoben, bis es zumindest teilweise sichtbar ist.

Rückgabewert Der Rückgabewert ist *LB_ERR*, wenn ein Fehler auftritt.

Beispiel Dieses Beispiel übergibt eine *LB_SETCARETINDEX*-Botschaft, um einem Element in einer Liste das fokussierende Rechteck zuzuordnen:

```
WPARAM wIndex;

wIndex = 0; /* Index auf das erste Element setzen */
SendDlgItemMessage(hdlg, ID_MYLISTBOX,
   LB_SETCARETINDEX, wIndex, 0L);
```

Siehe auch *LB_GETCARETINDEX*

STM_GETICON 3.1

Syntax

```
STM_GETICON

wParam = 0;         /* nicht verwendet, muß Null sein */
lParam = 0L;        /* nicht verwendet, muß Null sein */
```

Beschreibung Anwendungen übergeben die Botschaft *STM_SETICON*, um das Handle eines Symbols abzurufen, das einer Symbolressource zugeordnet ist.

Parameter Diese Botschaft verwendet keine Parameter.

Rückgabewert Der Rückgabewert ist das Handle des Symbols, wenn die Aktion erfolgreich war. Er ist Null, wenn ein Fehler auftritt oder dem Symbol keine Ressource zugeordnet ist.

Beispiel Dieses Beispiel ruft das Handle eines Symbols ab, das einer Symbolressource zugeordnet ist:

```
HICON hIcon;

hIcon = (HICON) SendDlgItemMessage(hdlg, IDD_ICON,
   STM_GETICON, 0, 0L);
```

Siehe auch *STM_SETICON*

STM_SETICON 3.1

Syntax

```
STM_SETICON

wParam = (WPARAM) (HICON) hicon; /* Handle des Symbols */
```

```
lParam = 0L;                    /* nicht verwendet,
                                   muß Null sein     */
```

Beschreibung Anwendungen übergeben die Botschaft *STM_SETICON*, um ein Symbol einer Symbolressource zuzuordnen.

Parameter *hicon:* Wert von *wParam*. Bezeichnet das Symbol, das der Symbolressource zugeordnet werden soll.

Rückgabewert Der Rückgabewert ist das Handle des Symbols, das zuvor der Symbolressource zugeordnet wurde. Er ist Null, wenn ein Fehler auftritt.

Beispiel Dieses Beispiel ordnet das systemdefinierte Fragezeichensymbol einer Symbolressource zu:

```
HICON hIcon, hOldIcon;

hIcon = LoadIcon((HANDLE) NULL, IDI_QUESTION);
hOldIcon = (HICON) SendDlgItemMessage(hdlg, IDD_ICON,
   STM_SETICON, hIcon, 0L);
```

Siehe auch *STM_GETICON*

WM_CHOOSEFONT_GETLOGFONT 3.1

Syntax

```
WM_CHOOSEFONT_GETLOGFONT

wParam = 0;                 /* nicht verwendet,
                               muß Null sein          */
lplf = (LPLOGFONT) lParam;  /* Adresse einer LOGFONT-
                               Struktur               */
```

Beschreibung Anwendungen übergeben eine *WM_CHOOSEFONT_GETLOGFONT*-Botschaft an das Dialogfenster *Schriftart*, das von der Funktion *ChooseFont* erstellt wurde, um die aktuelle *LOGFONT*-Struktur abzurufen.

Parameter *lplf:* Zeigt auf eine *LOGFONT*-Struktur, die Informationen über die aktuelle virtuelle Schrift aufnimmt.

Rückgabewert Diese Botschaft hat keinen Rückgabewert.

Hinweis Anwendungen benutzen diese Botschaft, um die *LOGFONT*-Struktur abzurufen, während das Dialogfenster *Schriftart* geöffnet ist. Wenn der Anwender das Dialogfenster schließt, erhält die Funktion *ChooseFont* Informationen über die *LOGFONT*-Struktur.

Siehe auch *WM_GETFONT*

WM_COMMNOTIFY 3.1

Syntax

```
WM_COMMNOTIFY

idDevice = wParam;                /* ID des seriellen
                                     Geräts           */
nNotifyStatus = LOWORD(lParam);   /* Benachrichtigungs-
                                     status-Flag      */
```

Beschreibung Die Botschaft *WM_COMMNOTIFY* wird von seriellen Gerätetreibern übergeben, wenn ein Ereignis an einer seriellen Schnittstelle (COM) auftritt. Die Botschaft zeigt den Status der Eingabe- oder Ausgabewarteschlange eines Fensters an.

Parameter *idDevice:* Wert von *wParam*. Bestimmt den Bezeichner des seriellen Geräts, das die Benachrichtigungsbotschaft übergibt.

nNotifyStatus: Wert des niederwertigen *WORD* von *lParam*. Enthält den Benachrichtigungsstatus im niederwertigen *WORD*. Der Benachrichtigungsstatus kann einer oder mehrere der folgenden Werte sein:

Wert	Bedeutung
CN_EVENT	Es ist ein Ereignis aufgetreten, das im Ereignis-*WORD* des seriellen Geräts aktiviert wurde. Dieses Ereignis wurde durch einen Aufruf der Funktion *SetCommEventMask* aktiviert. Die Anwendung sollte *GetCommEventMask* aufrufen, um zu ermitteln, welches Ereignis aufgetreten ist und um es zu löschen.
CN_RECEIVE	Es befinden sich mindestens *cbWriteNotify* Byte in der Eingabewarteschlange. Der Parameter *cbWriteNotify* ist ein Parameter der Funktion *EnableCommNotification*.
CN_TRANSMIT	Es befinden sich weniger als *cbOutQueue* Byte in der Ausgabewarteschlange. Der Parameter *cbOutQueue* ist ein Parameter der Funktion *EnableCommNotification*.

Rückgabewert Anwendungen sollten Null zurückgeben, wenn sie diese Botschaft bearbeiten.

Hinweis Diese Botschaft wird nur dann übergeben, wenn das Ereignis-*WORD* für das serielle Gerät verändert wird. Die übergebende Anwendung muß das Ereignis löschen, damit zukünftige Benachrichtigungen empfangen werden können.

Siehe auch *EnableCommNotification*

WM_DDE_ACK **2.x**

Syntax

```
#include <dde.h>
WM_DDE_ACK

wParam = (WPARAM) hwnd;           /* Handle des sen-
                                     denden Fensters  */
lParam = MAKELPARAM(wLow, wHigh); /* abhängig von der
                                     empfangenen
                                     Botschaft        */
```

Beschreibung

Die Botschaft *WM_DDE_ACK* benachrichtigt eine Anwendung über den Empfang und die Bearbeitung einer der Botschaften *WM_DDE_INITIATE*, *WM_DDE_EXECUTE*, *WM_DDE_DATA*, *WM_DDE_ADVISE*, *WM_DDE_UNADVISE* und *WM_DDE_POKE*. In einigen Fällen kann sie sich auch auf die Botschaft *WM_DDE_REQUEST* beziehen.

Parameter

hwnd: Wert von *wParam*. Bestimmt das Handle des Fensters, das die Botschaft sendet.

wLow: Wert des niederwertigen *WORD* von *lParam*. Die angegebenen Daten hängen davon ab, auf welche Botschaft *WM_DDE_ACK* antwortet:

Botschaft	Parameter	Beschreibung
WM_DDE_INITIATE	*aApplication*	Ein Atom, das den Namen der antwortenden Anwendung enthält.
WM_DDE_EXECUTE und alle anderen Botschaften	*wStatus*	Flags, die den Status der Antwort anzeigen.

wHigh: Wert des höherwertigen *WORD* von *lParam*. Die angegebenen Daten hängen davon ab, auf welche Botschaft *WM_DDE_ACK* antwortet:

Botschaft	Parameter	Beschreibung
WM_DDE_INITIATE	*aTopic*	Ein Atom, das das Thema enthält, dem das antwortende Server-Fenster zugeordnet ist.
WM_DDE_EXECUTE	*hCommands*	Ein Handle, das das Datenelement bezeichnet, das den Befehls-String enthält.
Alle anderen Botschaften	*aItem*	Ein Atom, das das Datenelement bestimmt, für das die Antwort gesendet wird.

Rückgabewert Diese Botschaft hat keinen Rückgabewert.

Hinweis Das *WORD wStatus* besteht aus einer *DDEACK*-Datenstruktur. Die *DDEACK*-Struktur hat folgende Form:

```
#include <dde.h>

typedef struct tagDDEACK { /* ddeack */
   WORD bAppReturnCode:8,
        reserved:6,
        fBusy:1,
        fAck:1;
} DDEACK;
```

Eine vollständige Beschreibung dieser Struktur finden Sie in Kapitel 7 »Strukturen«.

Senden
Außer als Antwort auf die Botschaft *WM_DDE_INITIATE* übergibt die Anwendung die Botschaft *WM_DDE_ACK* durch Aufruf der Funktion *PostMessage*, nicht durch *SendMessage*. Die Funktion *SendMessage* wird nur dann aufgerufen, wenn die Botschaft *WM_DDE_ACK* als Antwort auf *WM_DDE_INITIATE* gesendet werden soll.

Bei der Bestätigung einer Botschaft mit einem zugehörigen *aItem*-Atom kann die *WM_DDE_ACK*-sendende Anwendung das *aItem*-Atom der ursprünglichen Botschaft wiederverwenden. Sie kann das Atom jedoch auch löschen und ein neues erzeugen.

Bei der Bestätigung von *WM_DDE_EXECUTE* sollte die *WM_DDE_ACK*-sendende Anwendung das Objekt *hCommands* wiederverwenden, das der ursprünglichen *WM_DDE_EXECUTE*-Botschaft zugeordnet war.

Wenn eine Anwendung die Beendigung einer Konversation durch die Botschaft *WM_DDE_TERMINATE* eingeleitet hat und auf Zustimmung wartet, sollte die wartende Anwendung nachfolgende Botschaften von anderen Anwendungen weder positiv noch negativ bestätigen. Die wartende Anwendung sollte Atome oder gemeinsam benutzte Speicherobjekte löschen, die mit diesen Botschaften empfangen werden. Die Atome in einer Antwort auf die *WM_DDE_ACK*-Botschaft sollten nicht gelöscht werden.

Empfangen
Die *WM_DDE_ACK*-empfangende Anwendung sollte alle zugehörigen Atome löschen.

Empfängt eine Anwendung *WM_DDE_ACK* als Antwort auf eine Botschaft mit einem zugehörigen *hData*-Objekt, sollte sie dieses Objekt löschen.

Wenn die Anwendung eine negative *WM_DDE_ACK*-Botschaft als Antwort auf eine *WM_DDE_ADVISE*-Botschaft emp-

fängt, sollte sie das *hOptions*-Objekt löschen, das mit der ursprünglichen *WM_DDE_ADVISE*-Botschaft übergeben wurde.

Wenn eine Anwendung eine negative *WM_DDE_ACK*-Botschaft als Antwort auf eine *WM_DDE_EXECUTE*-Botschaft empfängt, sollte sie das *hCommands*-Objekt löschen, das mit der ursprünglichen *WM_DDE_EXECUTE*-Botschaft übergeben wurde.

Siehe auch *DDEACK, PostMessage, WM_DDE_ADVISE, WM_DDE_DATA, WM_DDE_EXECUTE, WM_DDE_INITIATE, WM_DDE_POKE, WM_DDE_REQUEST, WM_DDE_TERMINATE, WM_DDE_UNADVISE*

WM_DDE_ADVISE 2.x

Syntax

```
#include <dde.h>
WM_DDE_ADVISE

wParam = (WPARAM) hwnd;                 /* Handle des
                                           sendenden
                                           Fensters     */
lParam = MAKELPARAM(hOptions, aItem); /* übergabe-
                                           optionen und
                                           Datenelement */
```

Beschreibung Eine DDE-Client-Anwendung übergibt die Botschaft *WM_DDE_ADVISE* an eine DDE-Server-Anwendung, um die Aktualisierung eines Datenelements zu erhalten, sobald dieses verändert wird.

Parameter *hwnd:* Wert von *wParam*. Bezeichnet das sendende Fenster.

hOptions: Wert des niederwertigen *WORD* von *lParam*. Bestimmt das Handle eines globalen Speicherobjekts, das die Art der Datenübergabe bestimmt.

aItem: Wert des höherwertigen *WORD* von *lParam*. Bestimmt das angeforderte Datenelement.

Rückgabewert Diese Botschaft hat keinen Rückgabewert.

Hinweis Das durch den Parameter *hOptions* bezeichnete globale Speicherobjekt besteht aus einer *DDEADVISE*-Datenstruktur. Die *DDEADVISE*-Datenstruktur hat folgende Form:

```
#include <dde.h>

typedef struct tagDDEADVISE { /* ddeadv */
   WORD  reserved:14,
         fDeferUpd:1,
         fAckReq:1;
   short cfFormat;
} DDEADVISE;
```

Eine vollständige Beschreibung dieser Struktur finden Sie in Kapitel 7 »Strukturen«.

Unterstützt eine Anwendung mehr als ein Zwischenablageformat für ein einzelnes Thema und Element, kann sie mehrere *WM_DDE_ADVISE*-Botschaften für Thema und Element senden. Dabei wird in jeder Botschaft ein anderes Zwischenablageformat angegeben.

Senden
Die Anwendung übergibt die Botschaft *WM_DDE_ADVISE* durch Aufruf der Funktion *PostMessage*, nicht *SendMessage*.

Die Anwendung reserviert *hOptions* durch Aufruf der Funktion *GlobalAlloc* mit der Option *GMEM_DDESHARE*.

Die Anwendung reserviert *aItem* durch Aufruf der Funktion *GlobalAddAtom*.

Wenn die empfangende Server-Anwendung mit einer negativen *WM_DDE_ACK*-Botschaft antwortet, muß die sendende Client-Anwendung das *hOptions*-Objekt löschen.

Empfangen
Die Anwendung übergibt die Botschaft *WM_DDE_ACK*, um positiv oder negativ zu antworten. Beim Senden von *WM_DDE_ACK* kann die Anwendung das Atom *aItem* wiederverwenden oder löschen und ein neues erzeugen. Wenn die Botschaft *WM_DDE_ACK* positiv ist, sollte die Anwendung das *hOptions*-Objekt löschen. Andernfalls sollte die Anwendung das Objekt nicht löschen.

Siehe auch *DDEADVISE, GlobalAddAtom, GlobalAlloc, PostMessage, WM_DDE_DATA, WM_DDE_REQUEST*

WM_DDE_DATA 2.x

Syntax

```
#include <dde.h>
WM_DDE_DATA

wParam = (WPARAM) hwnd;              /* Handle des sen-
                                        denden Fensters */
lParam = MAKELPARAM(hData, aItem); /* Speicherobjekt und
                                        Datenelement    */
```

Beschreibung Eine DDE-Server-Anwendung übergibt eine *WM_DDE_DATA*-Botschaft an eine DDE-Client-Anwendung, um ein Datenelement an den Client zu übergeben oder um den Client von der Verfügbarkeit des Datenelements zu benachrichtigen.

Parameter *hwnd:* Wert von *wParam*. Bestimmt das Handle des Fensters, das die Botschaft sendet.

hData: Wert des niederwertigen *WORD* von *lParam*. Bezeichnet das globale Speicherobjekt, das die Daten und die zusätz-

lichen Informationen enthält. Das Handle sollte auf *NULL* gesetzt werden, wenn der Server den Client darüber benachrichtigt, daß der Wert des Datenelements während einer *warmen* Verbindung geändert wurde. Eine *warme* Verbindung wird eingerichtet, wenn der Client eine *WM_DDE_ADVISE*-Botschaft mit gesetzem Bit *fDeferUpd* sendet.

aItem: Wert des höherwertigen *WORD* von *lParam*. Bestimmt das Datenelement, für das Daten oder Benachrichtigungen gesendet werden.

Rückgabewert Diese Botschaft hat keinen Rückgabewert.

Hinweis Das durch den Parameter *hData* bezeichnete globale Speicherobjekt besteht aus einer *DDEDATA*-Struktur. Die *DDEDATA*-Struktur hat folgende Form:

```
#include <dde.h>

typedef struct tagDDEDATA { /* ddedat */
   WORD  unused:12,
         fResponse:1,
         fRelease:1,
         reserved:1,
         fAckReq:1;
   short cfFormat;
   BYTE  Value[1];
}  DDEDATA;
```

Eine vollständige Beschreibung dieser Struktur finden Sie in Kapitel 7 »Strukturen«.

Senden
Anwendungen senden die Botschaft *WM_DDE_DATA* durch Aufruf der Funktion *PostMessage*, nicht durch *SendMessage*.

Die Anwendung reserviert *hData* durch Aufruf der Funktion *GlobalAlloc* mit der Option *GMEM_DDESHARE*.

Die Funktion reserviert *aItem* durch Aufruf der Funktion *GlobalAddAtom*.

Wenn die empfangende Client-Anwendung mit einer negativen *WM_DDE_ACK*-Botschaft antwortet, muß die sendende Server-Anwendung das *hData*-Objekt löschen.

Setzt die sendende Server-Anwendung das Element *fRelease* der *DDEDATA*-Struktur auf *FALSE*, ist sie für das Löschen von *hData* verantwortlich, wenn sie eine positive oder negative Bestätigung empfängt.

Die Anwendung sollte niemals die beiden Elemente *fAckReq* und *fRelease* der *DDEDATA*-Struktur auf *FALSE* setzen. Wenn beide Elemente auf *FALSE* gesetzt sind, kann die sendende Server-Anwendung nicht ermitteln, wann *hData* gelöscht werden muß.

Empfangen
Wenn *fAckReq TRUE* ist, sendet die Anwendung die Botschaft *WM_DDE_ACK*, um positiv oder negativ zu antworten. Beim Senden von *WM_DDE_ACK* kann die Anwendung das Atom *aItem* wiederverwenden oder löschen und neu erzeugen.

Wenn *fAckReq FALSE* ist, löscht die Anwendung das Atom *aItem*.

Wenn die sendende Server-Anwendung für *hData NULL* angegeben hat, kann die empfangende Client-Anwendung die Übergabe der tatsächlichen Daten durch eine *WM_DDE_REQUEST*-Botschaft beim Server anfordern.

Nach der Bearbeitung einer *WM_DDE_DATA*-Botschaft, in der *hData* nicht *NULL* war, sollte die Anwendung *hData* löschen, wenn nicht eine der folgenden Bedingungen erfüllt ist:

- Das Element *fRelease* ist *FALSE*.
- Das Element *fRelease* ist *TRUE*, aber die empfangende Client-Anwendung antwortet mit einer negativen *WM_DDE_ACK*-Botschaft.

Siehe auch *DDEDATA, GlobalAddAtom, GlobalAlloc, PostMessage, WM_DDE_ACK, WM_DDE_ADVISE, WM_DDE_POKE, WM_DDE_REQUEST*

WM_DDE_EXECUTE 2.x

Syntax

```
#include <dde.h>
WM_DDE_EXECUTE

wParam = (WPARAM) hwnd;      /* Handle des sendenden
                                Fensters              */
lParam = MAKELPARAM(reserved, hCommands);
                             /* auszuführende
                                Befehle               */
```

Beschreibung Eine DDE-Client-Anwendung übergibt eine *WM_DDE_EXECUTE*-Botschaft an eine DDE-Server-Anwendung, um einen String an den Server zu übergeben, der als eine Reihe von Befehlen verarbeitet werden soll. Die Server-Anwendung sollte als Antwort eine *WM_DDE_ACK*-Botschaft senden.

Parameter *hwnd:* Wert von *wParam*. Bezeichnet das sendende Fenster.

reserved: Wert des niederwertigen *WORD* von *lParam*. Reserviert, muß Null sein.

hCommands: Wert des höherwertigen *WORD* von *lParam*. Bezeichnet ein globales Speicherobjekt, das die auszuführenden Befehle enthält.

Rückgabewert Diese Botschaft hat keinen Rückgabewert.

Hinweis

Der Befehls-String ist ein nullterminierter String. Er besteht aus einem oder mehreren Opcode-Strings, die in eckige Klammern eingeschlossen und durch Leerzeichen voneinander getrennt sind.

Jeder Opcode-String hat nachstehende Syntax, wobei die *parameters*-Liste optional ist.

```
opcode parameters
```

opcode ist ein einzelnes, anwendungsdefiniertes Symbol. Es darf keine Leerzeichen, Kommas, Klammern oder Anführungszeichen enthalten. Die Liste *parameters* kann beliebige anwendungsdefinierte Werte enthalten. Mehrere Parameter werden durch Kommas voneinander getrennt. Die gesamte Parameterliste wird in Klammern eingeschlossen. Parameter dürfen keine Kommas oder Klammern außerhalb eines String enthalten, der in Anführungszeichen steht. Wenn eine Klammer in einem String in Anführungszeichen erscheinen soll, muß sie doppelt geschrieben werden: »((«.

Die folgenden Beispiele sind gültige Befehls-Strings:

```
[connect][download(query1,results.txt)][disconnect]
[query("sales per employee for each district")]
[open("sample.xlm")][run("r1c1")]
```

Senden

Anwendungen senden die *WM_DDE_EXECUTE*-Botschaft durch Aufruf der Funktion *PostMessage*, nicht durch *SendMessage*.

Die Anwendung reserviert *hCommands* durch Aufruf der Funktion *GlobalAlloc* mit der Option *GMEM_DDESHARE*.

Die Anwendung, die die ursprüngliche *WM_DDE_EXECUTE*-Botschaft gesendet hat, muß das *hCommands*-Objekt löschen, das in der *WM_DDE_ACK*-Botschaft zurückgegeben wurde, wenn sie diese Botschaft als Anwort auf die *WM_DDE_EXECUTE*-Botschaft bearbeitet.

Empfangen

Die Anwendung sendet die *WM_DDE_ACK*-Botschaft, um positiv oder negativ zu antworten. Das *hCommands*-Objekt wird wiederverwendet.

Siehe auch

PostMessage, WM_DDE_ACK

WM_DDE_INITIATE 2.x

Syntax

```
#include <dde.h>
WM_DDE_INITIATE

wParam = (WPARAM) hwnd;        /* Handle des
                                  sendenden Fensters  */
```

```
lParam = MAKELPARAM(aApplication, aTopic);
                                    /* Anwendung und Thema */
```

Beschreibung Eine DDE-Client-Anwendung sendet eine *WM_DDE_INITIATE*-Botschaft, um eine Konversation mit Server-Anwendungen einzuleiten, die auf die angegebene Anwendung und die angegebenen Themennamen antworten.

Beim Empfang dieser Botschaft sollten alle Server-Anwendungen, deren Namen der Anwendung *aApplication* entspricht, und die das Thema *aTopic* unterstützen, diese Botschaft bestätigen (siehe Botschaft *WM_DDE_ACK*).

Parameter *hwnd:* Wert von *wParam*. Bezeichnet das sendende Fenster.

aApplication: Wert des niederwertigen *WORD* von *lParam*. Bestimmt den Namen der Anwendung, mit der eine Konversation eingerichtet werden soll. Der Anwendungsname darf keinen Schrägstrich (/) oder umgekehrten Schrägstrich (\) enthalten. Diese Zeichen sind für zukünftige Verwendungen in Netzimplementierungen reserviert. Wenn *aApplication NULL* ist, wird eine Konversation mit allen Anwendungen angefordert.

aTopic: Wert des höherwertigen *WORD* von *lParam*. Bestimmt das Thema, für das eine Konversation angefordert wird. Wenn das Thema *NULL* ist, wird eine Konversation für alle verfügbaren Themen angefordert.

Rückgabewert Diese Botschaft hat keinen Rückgabewert.

Hinweis Wenn *aApplication NULL* ist, kann jede Anwendung antworten. Ist *aTopic* Null, so ist jedes Thema gültig. Beim Empfang einer *WM_DDE_INITIATE*-Anforderung mit dem Wert *NULL* für den Parameter *aTopic* sollte eine Anwendung eine *WM_DDE_ACK*-Botschaft für jedes von ihr unterstützte Thema senden.

Senden
Die Anwendung sendet die Botschaft *WM_DDE_INITIATE* durch Aufruf der Funktion *SendMessage*, nicht durch *PostMessage*. Indem die Anwendung den ersten Parameter von *SendMessage* auf -1 setzt, wird die Botschaft an alle Fenster übergeben:

```
SendMessage(-1, WM_DDE_INITIATE, hwndClient,
   MAKELONG(aApp, aTopic));
```

Besitzt die Anwendung das Fenster-Handle des gewünschten Server bereits, kann sie *WM_DDE_INITIATE* direkt an das Server-Fenster senden. In diesem Fall wird das Fenster-Handle des Server als erster Parameter im Aufruf von *SendMessage* übergeben.

Die Anwendung reserviert *aApplication* und *aTopic* durch Aufruf von *GlobalAddAtom*.

Sobald *SendMessage* zurückkehrt, löscht die Anwendung die Atome *aApplication* und *aTopic*.

Empfangen
Um die Einrichtung einer Konversation abzuschließen, antwortet die Anwendung mit einer oder mehreren *WM_DDE_ACK*-Botschaften. Jede Botschaft steht für ein eigenes Thema. Beim Senden einer *WM_DDE_ACK*-Botschaft sollte die Anwendung neue Atome für *aApplication* und *aTopic* erzeugen. Die Atome, die mit der Botschaft *WM_DDE_INITIATE* übergeben wurden, sollten nicht wiederverwendet werden.

Siehe auch *GlobalAddAtom, SendMessage, WM_DDE_ACK*

WM_DDE_POKE 2.x

Syntax

```
#include <dde.h>
WM_DDE_POKE

wParam = (WPARAM) hwnd;            /* Handle des sen-
                                      denden Fensters */
lParam = MAKELPARAM(hData, aItem); /* Daten-Handle
                                      und Element     */
```

Beschreibung Eine DDE-Client-Anwendung übergibt eine *WM_DDE_POKE*-Botschaft an eine Server-Anwendung. Ein Client fordert mit dieser Botschaft den Server zur Entgegennahme eines nicht angeforderten Datenelements auf. Der Server sollte mit einer *WM_DDE_ACK*-Botschaft antworten, um mitzuteilen, ob er das Datenelement entgegennimmt.

Parameter *hwnd:* Wert von *wParam*. Bestimmt das Handle des Fensters, das die Botschaft sendet.

hData: Wert des niederwertigen *WORD* von *lParam*. Bezeichnet die Daten, die gesendet werden. Das Handle bezeichnet ein globales Speicherobjekt, das eine *DDEPOKE*-Datenstruktur enthält. Die *DDEPOKE*-Struktur hat folgende Form:

```
#include <dde.h>

typedef struct tagDDEPOKE { /* ddepok */
   WORD   unused:13,
          fRelease:1,
          fReserved:2,
   short  cfFormat;
   BYTE   Value[1];
} DDEPOKE;
```

Eine vollständige Beschreibung dieser Struktur finden Sie im Kapitel 7 »Strukturen«.

aItem: Wert des höherwertigen *WORD* von *lParam*. Bestimmt ein globales Atom, das das Datenelement bezeichnet, welches an den Server übergeben werden soll.

Rückgabewert Diese Botschaft hat keinen Rückgabewert.

Hinweis **Senden**
Die sendende Client-Anwendung sollte folgendes tun:

- Die Funktion *PostMessage* aufrufen, um die Botschaft *WM_DDE_POKE* zu senden.
- Die Funktion *GlobalAlloc* mit der Option *GMEM_DDESHARE* aufrufen, um Speicher für die Daten zu reservieren.
- Die Funktion *GlobalAddAtom* aufrufen, um ein Atom für das Datenelement zu erzeugen.
- Das globale Speicherobjekt löschen, wenn die Server-Anwendung mit einer negativen *WM_DDE_ACK*-Botschaft antwortet.
- Das globale Speicherobjekt löschen, wenn der Client das Element *fRelease* der *DDEPOKE*-Struktur auf *FALSE* gesetzt hat und der Server mit einer positiven oder negativen *WM_DDE_ACK*-Botschaft antwortet.

Empfangen
Die empfangende Server-Anwendung sollte folgendes tun:

- Durch das Senden einer *WM_DDE_ACK*-Botschaft positiv oder negativ antworten. Beim Senden dieser Botschaft kann sie das Atom des Datenelements wiederverwenden oder löschen und ein neues erzeugen.
- Das globale Speicherobjekt löschen, nachdem die Botschaft *WM_DDE_POKE* bearbeitet wurde. Dies gilt nicht, wenn das Flag *fRelease* auf *FALSE* gesetzt war. Es gilt auch dann nicht, wenn das Flag *fRelease* auf *TRUE* gesetzt war, der Server aber mit einer negativen *WM_DDE_ACK*-Botschaft geantwortet hat.

Siehe auch *DDEPOKE, GlobalAlloc, PostMessage, WM_DDE_ACK, WM_DDE_DATA*

WM_DDE_REQUEST 2.x

Syntax
```
#include <dde.h>
WM_DDE_REQUEST

wParam = (WPARAM) hwnd;          /* Handle des
                                    sendenden Fensters */
```

```
lParam = MAKELPARAM(cfFormat, aItem);
                                    /* Zwischenablage-
                                       format und Element */
```

Beschreibung Eine DDE-Client-Anwendung sendet die *WM_DDE_REQUEST*-Botschaft an eine DDE-Server-Anwendung, um den Wert eines Datenelements abzufragen.

Parameter *hwnd:* Wert von *wParam*. Bezeichnet das sendende Fenster.

cfFormat: Wert des niederwertigen *WORD* von *lParam*. Bestimmt die Nummer eines Standardformats oder eines registrierten Formats der Zwischenablage.

aItem: Wert des höherwertigen *WORD* von *lParam*. Bestimmt, welches Datenelement beim Server angefordert wird.

Rückgabewert Diese Botschaft hat keinen Rückgabewert.

Hinweis **Senden**
Die Anwendung sendet die *WM_DDE_REQUEST*-Botschaft durch Aufruf der Funktion *PostMessage*, nicht durch *SendMessage*.

Die Anwendung reserviert *aItem* durch Aufruf der Funktion *GlobalAddAtom*.

Empfangen
Wenn die empfangende Server-Anwendung die Anforderung erfüllen kann, antwortet sie mit einer *WM_DDE_DATA*-Botschaft, die die angeforderten Daten enthält. Andernfalls antwortet sie mit einer negativen *WM_DDE_ACK*-Botschaft.

Die Anwendung kann das Atom *aItem* wiederverwenden oder es löschen und ein neues erzeugen, wenn sie mit den Botschaften *WM_DDE_DATA* oder *WM_DDE_ACK* antwortet.

Siehe auch *GlobalAddAtom, PostMessage, WM_DDE_ACK*

WM_DDE_TERMINATE 2.x

Syntax

```
#include <dde.h>
WM_DDE_TERMINATE

wParam = (WPARAM) hwnd;     /* Handle des sendenden
                               Fensters              */
lParam = 0L;                /* nicht verwendet,
                               muß Null sein.        */
```

Beschreibung Eine DDE-Anwendung (Client oder Server) sendet eine *WM_DDE_TERMINATE*-Botschaft, um eine Konversation zu beenden.

Parameter *hwnd:* Wert von *wParam*. Bezeichnet das sendende Fenster.

Rückgabewert Diese Botschaft hat keinen Rückgabewert.

Hinweis

Senden
Die Anwendung sendet die *WM_DDE_TERMINATE*-Botschaft durch Aufruf der Funktion *PostMessage*, nicht durch *SendMessage*.

Die sendende Anwendung sollte keine Botschaften beantworten, die von der empfangenden Anwendung gesendet werden, während sie noch auf die Bestätigung der Beendigung wartet. Wenn die sendende Anwendung andere Botschaften als *WM_DDE_TERMINATE* von der empfangenden Anwendung erhält, sollte sie alle Atome oder gemeinsam benutzten Speicherobjekte löschen, die diese Botschaften begleiten.

Empfangen
Die Anwendung antwortet durch Senden einer *WM_DDE_TERMINATE*-Botschaft.

Siehe auch *PostMessage*

WM_DDE_UNADVISE 2.x

Syntax

```
#include <dde.h>
WM_DDE_UNADVISE

WParam = (WPARAM) hwnd;          /* Handle des
                                    sendenden Fensters  */
lParam = MAKELPARAM(cfFormat, aItem);
                                 /* Zwischenablageformat
                                    und Element         */
```

Beschreibung Eine DDE-Client-Anwendung übergibt eine *WM_DDE_UNADVISE*-Botschaft, um eine Server-Anwendung darüber zu informieren, daß das angegebene Datenelement oder das Zwischenablageformat für das Element nicht länger aktualisiert werden soll. Damit wird die *warme* bzw. *heiße* Verbindung für das angegebene Element beendet.

Parameter *hwnd:* Wert von *wParam*. Bezeichnet das sendende Fenster.

cfFormat: Wert des niederwertigen *WORD* von *lParam*. Bestimmt das Zwischenablageformat des Elements, für das die Aktualisierung beendet werden soll.

Wenn der Parameter *cfFormat NULL* ist, werden alle aktiven *WM_DDE_ADVISE*-Konversationen für dieses Element beendet.

aItem: Wert des höherwertigen *WORD* von *lParam*. Bestimmt das Element, für das die Aktualisierung beendet werden soll. Wenn *aItem NULL* ist, werden alle aktiven *WM_DDE_ADVISE*-Konversationen dieses Client beendet.

Rückgabewert Diese Botschaft hat keinen Rückgabewert.

Hinweis

Senden
Die Anwendung sendet die Botschaft *WM_DDE_UNADVISE* durch Aufruf der Funktion *PostMessage*, nicht durch *SendMessage*. Die Anwendung reserviert *aItem* durch Aufruf von *GlobalAddAtom*.

Empfangen
Die Anwendung sendet die Botschaft *WM_DDE_ACK*, um positiv oder negativ zu antworten. Die Anwendung kann beim Senden von *WM_DDE_ACK* das Atom *aItem* wiederverwenden oder es löschen und ein neues erzeugen.

Siehe auch *GlobalAddAtom, PostMessage, WM_DDE_ACK*

WM_DROPFILES 3.1

Syntax

```
WM_DROPFILES

hDrop = (HANDLE) wParam;    /* Handle der internen
                               Struktur für abgelegte
                               Dateien              */
```

Beschreibung Die Botschaft *WM_DROPFILES* wird gesendet, wenn der Benutzer die linke Maustaste über einem Fenster einer Anwendung losläßt, die sich selbst als Empfänger abgelegter Dateien registriert hat.

Parameter *hDrop:* Wert von *wParam*. Bezeichnet eine interne Datenstruktur, die die abgelegten Dateien beschreibt. Dieses Handle wird von den Funktionen *DragFinish, DragQueryFile* und *DragQueryPoint* benutzt, um Informationen über die abgelegten Dateien abzurufen.

Rückgabewert Anwendungen sollten Null zurückgeben, wenn sie diese Botschaft bearbeiten.

Siehe auch *DragAcceptFiles, DragFinish, DragQueryFile, DragQueryPoint*

WM_PALETTEISCHANGING 3.1

Syntax

```
WM_PALETTEISCHANGING

hwndRealize = (HWND) wParam; /* Handle des Fensters,
                                das die Palette
                                realisieren soll     */
```

Beschreibung Die Botschaft *WM_PALETTEISCHANGING* informiert Anwendungen darüber, daß eine Anwendung ihre virtuelle Palette realisieren wird.

Parameter *hwndRealize:* Wert von *wParam*. Bestimmt das Handle des Fensters, das seine virtuelle Palette realisieren wird.

Rückgabewert Eine Anwendung sollte Null zurückgeben, wenn sie diese Botschaft bearbeitet.

Siehe auch *WM_PALETTECHANGED, WM_QUERYNEWPALETTE*

WM_POWER 3.1

Syntax

```
WM_POWER

fwPowerEvt = wParam;     /* Benachrichtigungsbotschaft */
```

Beschreibung Die Botschaft *WM_POWER* wird gesendet, wenn das System (normalerweise ein batteriebetriebener PC) in den Stromsparmodus umschalten wird.

Parameter *fwPowerEvt:* Wert von *wParam*. Bestimmt eine Benachrichtigungsbotschaft. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
PWR_SUSPEND-REQUEST	Das System wird den Stromsparmodus aktivieren.
PWR_SUSPEND-RESUME	Das System beendet den Stromsparmodus. Bevor der Stromsparmodus aktiviert wurde, hat das System die Benachrichtigung *PWR_SUSPENDREQUEST* an die Anwendung gesendet. Anwendungen sollten die nötigen Operationen durchführen, um ihre Ausführung wieder aufzunehmen.
PWR_CRITICAL-RESUME	Das System beendet den Stromsparmodus. Der Stromsparmodus wurde aktiviert, ohne daß die Anwendung vorher die Benachrichtigungsbotschaft *PWR_SUSPENDREQUEST* erhalten hat. Anwendungen sollten alle nötigen Operationen zur Wiederaufnahme ihrer Ausführung durchführen.

Rückgabewert Der von einer Anwendung zurückzugebende Wert hängt wie folgt vom Parameter *wParam* ab:

Wert von *wParam*	Rückgabewert
PWR_SUSPEND-REQUEST	*PWR_FAIL* verhindert die Umschaltung des Systems in den Stromsparmodus. *PWR_OK* läßt die Umschaltung zu.
PWR_SUSPEND-RESUME	0
PWR_CRITICAL-RESUME	0

Hinweis Diese Botschaft wird nur an Anwendungen in Systemen übergeben, die ein APM-BIOS besitzen (Advanced Power Management). Sie wird vom Treiber für die Überwachung der Stromversorgung an jedes Fenster übergeben, das von der Funktion *EnumWindows* zurückgegeben wird.

Im Stromsparmodus wird am meisten Strom gespart, obwohl alle Daten und Parameter gesichert bleiben. Die Inhalte des RAM (Random Access Memory) bleiben erhalten, die meisten Geräte werden ausgeschaltet.

Siehe auch *EnumWindows*

WM_QUEUESYNC 3.1

Syntax `WM_QUEUESYNC`

Beschreibung Die Botschaft *WM_QUEUESYNC* wird von einer CBT-Anwendung (computer-based training) gesendet, um die Botschaften durch Benutzereingaben von Botschaften zu unterscheiden, die durch einen Wiedergabeprotokollfilter gesendet werden (*WH_JOURNALPLAYBACK*).

Parameter Diese Botschaft verwendet keine Parameter.

Rückgabewert Eine CBT-Anwendung sollte Null zurückgeben, wenn sie diese Botschaft bearbeitet.

Hinweis Wenn eine CBT-Anwendung einen Wiedergabeprotokollfilter benutzt, ist die erste und die letzte zurückgegebene Botschaft *WM_QUEUESYNC*. Auf diese Weise kann eine CBT-Anwendung die Botschaften durch Benutzereingaben untersuchen, ohne dies für Ereignisse tun zu müssen, die sie selber gesendet hat.

WM_SYSTEMERROR 3.1

Syntax `WM_SYSTEMERROR`

```
wErrSpec = wParam;      /* Zeitpunkt des Fehlers      */
```

Beschreibung Die Botschaft *WM_SYSTEMERROR* wird gesendet, wenn der Windows-Kernel einen Fehler feststellt, das Meldungsfenster für Systemfehler aber nicht darstellen kann.

Parameter *wErrSpec:* Wert von *wParam*. Bestimmt, wann der Fehler aufgetreten ist. Zur Zeit ist nur der Wert 1 gültig. Er zeigt an, daß der Fehler beim Beenden einer Task oder Bibliothek aufgetreten ist.

Rückgabewert Eine Anwendung sollte Null zurückgeben, wenn sie diese Botschaft bearbeitet.

Hinweis Eine Shell-Anwendung sollte diese Botschaft bearbeiten. Sie sollte ein Meldungsfenster darstellen, das anzeigt, daß ein Fehler aufgetreten ist.

WM_USER 2.x

Syntax `WM_USER`

Beschreibung *WM_USER* ist eine Konstante, die Anwendungen beim Definieren privater Botschaften unterstützt.

Hinweis Die Konstante *WM_USER* wird verwendet, um zwischen Botschaftswerten zu unterscheiden, die für Windows reserviert sind und anderen Werten, die von einer Anwendung zum Senden von Botschaften innerhalb einer privaten Fensterklasse benutzt werden können. Es gibt vier Bereiche für Botschaftsnummern:

Bereich	Bedeutung
0 bis WM_USER -1	Botschaften, die für die Benutzung durch Windows reserviert sind.
WM_USER bis 0x7FFF	Integer-Botschaften, die in privaten Fensterklassen benutzt werden können.
0x8000 bis 0xBFFF	Botschaften, die für die Benutzung durch Windows reserviert sind.
0xC000 bis 0xFFFF	String-Botschaften, die von Anwendungen benutzt werden können.

Botschaftsnummern im ersten Bereich (0 bis *WM_USER* -1) sind von Windows definiert. Alle Werte in diesem Bereich, die nicht explizit definiert sind, sind für eine zukünftige Verwendung durch Windows reserviert. Dieses Kapitel beschreibt die Botschaften in diesem Bereich.

Botschaftsnummern im zweiten Bereich (*WM_USER* bis 0x7FFF) können von Anwendungen definiert und benutzt werden, um Botschaften innerhalb einer privaten Fensterklasse zu senden. Diese Werte dürfen nicht benutzt werden, um Botschaften zu definieren, die auch außerhalb der Anwendung von Bedeutung sind, da einige vordefinierte Fensterklassen bereits Werte in diesem Bereich definiert haben. Vordefinierte Steuerelementklassen wie *BUTTON*, *EDIT*, *LISTBOX* und *COMBOBOX* können diese Werte benutzen. Botschaften in diesem Bereich sollten nicht an andere Anwendungen übergeben werden, wenn diese nicht zum Austausch von Botschaften entworfen wurden und den Botschaftsnummern die gleiche Bedeutung zugeordnet haben.

Botschaftsnummern im dritten Bereich (0x8000 bis 0xBFFF)

sind für eine zukünftige Verwendung durch Windows reserviert.

Botschaftsnummern im vierten Bereich (0xC000 bis 0xFFFF) werden zur Laufzeit definiert, wenn eine Anwendung die Funktion *RegisterWindowMessage* aufruft, um eine Botschaftsnummer für einen String zu erhalten. Alle Anwendungen, die denselben String registrieren, können die zugeordnete Botschaftsnummer zum Austausch von Botschaften verwenden. Die tatsächliche Botschaftsnummer ist keine Konstante und kann in unterschiedlichen Windows-Arbeitssitzungen unterschiedlich sein.

Siehe auch *RegisterWindowMessage*

WM_WINDOWPOSCHANGED 3.1

Syntax

```
WM_WINDOWPOSCHANGED

pwp = (const WINDOWPOS FAR*) lParam;  /* Adresse der
                                         Struktur     */
```

Beschreibung Die Botschaft *WM_WINDOWPOSCHANGED* wird an Fenster übergeben, deren Größe, Position oder Z-Ordnung infolge eines Aufrufs der Funktion *SetWindowPos* oder einer anderen Fensterverwaltungsfunktion geändert wurde.

Parameter *pwp:* Wert von *lParam*. Zeigt auf eine *WINDOWPOS*-Datenstruktur, die Informationen über die neue Größe und Position des Fensters enthält. Die *WINDOWPOS*-Struktur hat folgende Form:

```
typedef struct tagWINDOWPOS { /* wp */
   HWND hwnd;
   HWND hwndInsertAfter;
   int x;
   int y;
   int cx;
   int cy;
   UINT flags;
} WINDOWPOS;
```

Rückgabewert Eine Anwendung sollte Null zurückgeben, wenn sie diese Botschaft bearbeitet.

Hinweis Die Funktion *DefWindowProc* sendet die Botschaften *WM_SIZE* und *WM_MOVE* an das Fenster, wenn sie die Botschaft *WM_WINDOWPOSCHANGED* bearbeitet. Diese Botschaften werden nicht gesendet, wenn eine Anwendung die Botschaft *WM_WINDOWPOSCHANGED* bearbeitet, ohne *DefWindowProc* aufzurufen. Die Durchführung von Verschiebungen oder Größenänderungen während der Bearbeitung der Botschaft

WM_WINDOWPOSCHANGED verläuft schneller, wenn *DefWindowProc* nicht aufgerufen wird.

Siehe auch *WM_MOVE, WM_SIZE, WM_WINDOWPOSCHANGING*

WM_WINDOWPOSCHANGING 3.1

Syntax

```
WM_WINDOWPOSCHANGING

pwp = (WINDOWPOS FAR*) lParam;  /* Adresse der
                                   WINDOWPOS-Struktur */
```

Beschreibung Die *WM_WINDOWPOSCHANGING*-Botschaft wird an ein Fenster gesendet, dessen Größe, Position oder Z-Ordnung sich als Ergebnis eines Aufrufs der Funktion *SetWindowPos* oder einer anderen Fensterverwaltungsfunktion ändern soll.

Parameter *pwp:* Wert von *lParam*. Zeigt auf eine *WINDOWPOS*-Datenstruktur, die Informationen über die neue Größe und Position des Fensters enthält. Die *WINDOWPOS*-Struktur hat folgende Form:

```
typedef struct tagWINDOWPOS { /* wp */
   HWND hwnd;
   HWND hwndInsertAfter;
   int x;
   int y;
   int cx;
   int cy;
   UINT flags;
} WINDOWPOS;
```

Rückgabewert Eine Anwendung sollte Null zurückgeben, wenn sie diese Botschaft bearbeitet.

Hinweis Während der Bearbeitung dieser Botschaft beeinflußt die Veränderung der entsprechenden Werte in der *WINDOWPOS*-Struktur die neue Größe, Position oder Z-Ordnung des Fensters. Eine Anwendung kann diese Veränderungen verhindern, indem sie die entsprechenden Bits im Element *flags* der *WINDOWPOS*-Struktur setzt oder löscht.

Wenn ein Fenster den Stil *WS_OVERLAPPED* oder *WS_THICKFRAME* hat, bearbeitet die Funktion *DefWindowProc* eine *WM_WINDOWPOSCHANGING*-Botschaft durch das Senden einer *WM_GETMINMAXINFO*-Botschaft an dieses Fenster. Dies geschieht, um die neue Größe und Position des Fensters zu überprüfen und die Stile *CS_BYTEALIGNCLIENT* und *CS_BYTEALIGN* zuzuordnen. Anwendungen können dies unterdrücken, indem sie die Botschaft *WM_WINDOWPOSCHANGING* nicht an die Funktion *DefWindowProc* übergeben.

Siehe auch *WM_WINDOWPOSCHANGED*

Benachrichtigungs-botschaften

BN_HILITE 2.x

Syntax `BN_HILITE`

Beschreibung Die Benachrichtigungsbotschaft *BN_HILITE* wird gesendet, wenn der Benutzer einen Schalter hervorhebt. Diese Benachrichtigung dient der Kompatibilität mit Anwendungen, die für Windows-Versionen vor Windows 3.0 geschrieben wurden. Neue Anwendungen sollten den Schalterstil *BS_OWNERDRAW* und die *DRAWITEMSTRUCT*-Struktur für diese Aufgabe verwenden.

Siehe auch *DRAWITEMSTRUCT*, *WM_DRAWITEM*

BN_PAINT 2.x

Syntax `BN_PAINT`

Beschreibung Die Benachrichtigungsbotschaft *BN_PAINT* wird gesendet, wenn ein Schalter gezeichnet werden sollte. Diese Benachrichtigung dient der Kompatibilität mit Anwendungen, die für Windows-Versionen vor der Version 3.0 geschrieben wurden. Neue Anwendungen sollten den Schalterstil *BS_OWNERDRAW* und die *DRAWITEMSTRUCT*-Struktur für diese Aufgabe verwenden.

Siehe auch *DRAWITEMSTRUCT*, *WM_DRAWITEM*

BN_UNHILITE 2.x

Syntax `BN_UNHILITE`

Beschreibung Die Benachrichtigungsbotschaft *BN_UNHILITE* wird gesendet, wenn die Hervorhebung eines Schalters entfernt werden soll. Diese Benachrichtigung dient der Kompatibilität mit Anwendungen, die für Windows-Versionen vor der Version 3.0 geschrieben wurden. Neue Anwendungen sollten den Schalterstil *BS_OWNERDRAW* und die *DRAWITEMSTRUCT*-Struktur für diese Aufgabe verwenden.

Siehe auch *DRAWITEMSTRUCT*, *WM_DRAWITEM*

CBN_CLOSEUP 3.1

Beschreibung Die Benachrichtigungsbotschaft *CBN_CLOSEUP* wird gesendet, wenn die Liste eines Kombinationsfensters verborgen

wird. Das übergeordnete Fenster dieses Steuerelements erhält diese Benachrichtigungsbotschaft durch eine *WM_COMMAND*-Botschaft.

Parameter *wParam:* Bestimmt den Bezeichner des Kombinationsfensters.

lParam: Bestimmt das Handle des Kombinationsfensters im niederwerten *WORD* und die Benachrichtigungsbotschaft *CBN_CLOSEUP* im höherwertigen.

Hinweis Diese Benachrichtigungsbotschaft wird nicht übergeben, wenn das Kombinationsfenster den Stil *CBS_SIMPLE* hat.

Die Reihenfolge, in der die Benachrichtigungen gesendet werden, kann nicht beeinflußt werden. Die Benachrichtigung *CBN_SELSCHANGE* kann vor oder nach der Benachrichtigung *CBN_CLOSEUP* auftreten.

Siehe auch *CBN_DROPDOWN, CBN_SELCHANGE, WM_COMMAND*

CBN_SELENDCANCEL 3.1

Beschreibung Die Benachrichtigungsbotschaft *CBN_SELENDCANCEL* wird gesendet, wenn der Benutzer ein Element und dann ein anderes Fenster oder Steuerelement anklickt, um die Liste des Kombinationsfensters zu verbergen. Diese Benachrichtigungsbotschaft wird vor der Benachrichtigungsbotschaft *CBN_CLOSEUP* gesendet, damit die Auswahl des Benutzers ignoriert wird.

Parameter *wParam:* Bestimmt den Bezeichner des Kombinationsfensters.

lParam: Bestimmt das Handle des Kombinationsfensters im niederwertigen *WORD*. Das höherwertige *WORD* enthält die Benachrichtigungsbotschaft *CBN_SELENDCANCEL*.

Hinweis Die Benachrichtigungsbotschaften *CBN_SELENDCANCEL* oder *CBN_SELENDOK* werden gesendet, auch wenn die Benachrichtigungsbotschaft *CBN_CLOSEUP* nicht gesendet wird. Letzteres ist z.B. dann der Fall, wenn das Kombinationsfenster den Stil *CBS_SIMPLE* hat.

Siehe auch *CBN_SELENDOK, WM_COMMAND*

CBN_SELENDOK 3.1

Beschreibung Die Benachrichtigungsbotschaft *CBN_SELENDOK* wird gesendet, wenn der Benutzer ein Element auswählt und dann die Taste CR drückt oder den nach unten weisenden Pfeil anklickt, um die Liste des Kombinationsfensters zu verbergen. Diese Benachrichtigungsbotschaft wird vor der Benachrichtigungs-

botschaft *CBN_CLOSEUP* gesendet, damit die Auswahl des Benutzers berücksichtigt wird.

Parameter *wParam:* Bestimmt den Bezeichner des Kombinationsfensters.

lParam: Bestimmt das Handle des Kombinationsfensters im niederwertigen *WORD*. Im höherwertigen *WORD* befindet sich die Benachrichtigungsbotschaft *CBN_SELENDOK*.

Hinweis Die Benachrichtigungsbotschaften *CBN_SELENDOK* oder *CBN_SELENDCANCEL* werden auch dann gesendet, wenn die Benachrichtigungsbotschaft *CBN_CLOSEUP* nicht gesendet wird. Letzteres ist der Fall, wenn ein Kombinationsfenster den Stil *CBS_SIMPLE* hat.

Siehe auch *CBN_SELENDCANCEL, WM_COMMAND*

LBN_SELCANCEL 3.1

Syntax `LBN_SELCANCEL`

Beschreibung Die Benachrichtigungsbotschaft *LBN_SELCANCEL* wird gesendet, wenn der Benutzer die Auswahl in einer Liste abbricht. Das übergeordnete Fenster der Liste erhält diese Benachrichtigungsbotschaft durch eine *WM_COMMAND*-Botschaft.

Parameter *wParam:* Bestimmt den Bezeichner der Liste.

lParam: Bestimmt das Handle der Liste im niederwertigen *WORD*. Das höherwertige *WORD* enthält die Benachrichtigungsbotschaft *LBN_SELCANCEL*.

Hinweis Diese Benachrichtigung kann nur für Listen verwendet werden, die den Stil *LBS_NOTIFY* haben.

Siehe auch *LBN_DBLCLK, LBN_SELCHANGE, LB_SETCURSEL, WM_COMMAND*

7. Strukturen

ABC 3.1

Definition

```
typedef struct tagABC {   /* abc */
   int abcA;
   UINT abcB;
   int abcC;
} ABC;
```

Beschreibung Die *ABC*-Struktur enthält die Breite eines Zeichens in einer TrueType-Schrift.

Elemente *abcA:* Bestimmt den »A«-Breitenanteil des Zeichens. Der »A«-Breitenanteil ist die Entfernung, die zur aktuellen Position addiert wird, bevor der Zeichenkörper gesetzt wird.

abcB: Bestimmt den »B«-Breitenanteil des Zeichens. Der »B«-Breitenanteil ist die Breite des Zeichenkörpers.

abcC: Bestimmt den »C«-Breitenanteil des Zeichens. Der »C«-Breitenanteil ist die Entfernung, die zur aktuellen Position addiert wird, nachdem der Zeichenkörper gesetzt wurde.

Hinweis Die gesamte Breite eines Zeichens ist die Summe der »A«-, »B«- und »C«-Breitenanteile. Die Werte von »A« und »C« können negativ werden, wenn das Zeichen links oder rechts Unterschneidungen aufweist.

Siehe auch *GetCharABCWidths*

CBT_CREATEWND 3.1

Definition

```
typedef struct tagCBT_CREATEWND {  /* cbtcw  */
   CREATESTRUCT FAR* lpcs;
   HWND hwndInsertAfter;
} CBT_CREATEWND;
```

Beschreibung Die *CBT_CREATEWND*-Struktur enthält Informationen, die an eine *WH_CBT*-Filterfunktion übergeben werden, bevor ein Fenster erzeugt wird.

Elemente *lpcs:* Zeigt auf eine *CREATESTRUCT*-Struktur, die die Initialisierungsparameter für das zu erzeugende Fenster enthält.

hwndInsertAfter: Bezeichnet das Fenster in der Liste der Fensterverwaltung, das vor dem zu erzeugenden Fenster steht. Wenn dieser Parameter *NULL* ist, ist das zu erzeugende Fenster das oberste in der Liste. Ist der Parameter 1, wird das neue Fenster das unterste in der Liste.

Siehe auch *CBTProc*, *SetWindowsHook*

CBTACTIVATESTRUCT 3.1

Definition
```
typedef struct tagCBTACTIVATESTRUCT { /* cas */
   BOOL fMouse;
   HWND hWndActive;
} CBTACTIVATESTRUCT;
```

Beschreibung Die *CBTACTIVATESTRUCT*-Struktur enthält Informationen, die an eine *WH_CBT*-Filterfunktion übergeben werden, bevor ein Fenster aktiviert wird.

Elemente *fMouse:* Gibt an, ob das Fenster aufgrund des Klickens mit der Maus aktiviert wird. Dieser Wert ist ungleich Null, falls ein Mausklick das Fenster aktiviert. Andernfalls ist er Null.

hWndActive: Bezeichnet das aktuell aktive Fenster.

Siehe auch *SetWindowsHook*

CHOOSECOLOR 3.1

Definition
```
#include <commdlg.h>
typedef struct tagCHOOSECOLOR {  /* cc */
  DWORD lStructSize;
  HWND hwndOwner;
  HWND hInstance;
  COLORREF rgbResult;
  COLORREF FAR*  lpCustColors;
  DWORD Flags;
  LPARAM lCustData;
  UINT (CALLBACK* lpfnHook) (HWND, UINT, WPARAM, LPARAM);
  LPCSTR lpTemplateName;
} CHOOSECOLOR;
```

Beschreibung Die *CHOOSECOLOR*-Struktur enthält Informationen, die das System zum Initialisieren des systemdefinierten Dialogfensters *Farben* benutzt. Nachdem der Benutzer den Schalter *OK* angewählt hat, um das Dialogfenster zu schließen, liefert das System Informationen über die Auswahl des Benutzers in dieser Struktur zurück.

Elemente

lStructSize: Bestimmt die Länge der Struktur in Byte. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

hwndOwner: Bezeichnet das Fenster, zu dem das Dialogfenster gehört. Dieses Element kann jedes gültige Handle eines Fensters sein. Wenn das Dialogfenster keinem Fenster zugeordnet werden soll, sollte das Element *NULL* sein.

Ist das Flag *CC_SHOWHELP* gesetzt, muß *hwndOwner* das Fenster bezeichnen, zu dem das Dialogfenster gehört. Die Fensterfunktion des besitzenden Fensters erhält eine Benachrichtigungsbotschaft, wenn der Benutzer den *Hilfe*-Schalter anwählt. Der Bezeichner der Benachrichtigungsbotschaft ist der Wert, den die Funktion *RegisterWindowMessage* liefert, wenn ihr *HELPMSGSTRING* als Argument übergeben wird. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

hInstance: Bezeichnet einen Datenblock, der die Schablone des Dialogfensters enthält, die durch das Element *lpTemplateName* bestimmt ist. Dieses Element wird nur benutzt, wenn das Element *Flags* das Flag *CC_ENABLETEMPLATE* oder *CC_ENABLETEMPLATEHANDLE* angibt. Andernfalls wird dieses Element ignoriert. Das Element wird für die Initialisierung des Dialogfensters benötigt.

rgbResult: Bestimmt die Farbe, die ausgewählt ist, wenn das Fenster angezeigt wird. Es gibt die Farbwahl des Benutzers an, nachdem dieser den Schalter *OK* angewählt hat, um das Dialogfenster zu schließen. Falls das Flag *CC_RGBINIT* im Element *Flags* gesetzt ist, bevor das Dialogfenster angezeigt wird und sich der Wert dieses Elements nicht unter den verfügbaren Farben befindet, wählt das System die ähnlichste verfügbare Grundfarbe. Ist das Element *NULL*, ist Schwarz die erste ausgewählte Farbe. Dieses Element wird für die Initialisierung des Dialogfensters benötigt und beim Schließen des Fensters gesetzt.

lpCustColors: Zeigt auf ein Array mit 16 *DWORD*-Werten. Jeder Wert gibt die Intensitäten der roten, grünen und blauen Farbkomponenten (RGB) eines benutzerdefinierten Farbfelds im Dialogfenster an. Wenn ein Benutzer eine Farbe verändert, aktualisiert das System das Array mit den neuen RGB-Werten. Dieses Element wird für die Initialisierung des Dialogfensters benötigt und beim Schließen des Fensters gesetzt.

Flags: Gibt die Initialisierungs-Flags des Dialogfensters an. Dieses Element kann eine Kombination der folgenden Werte sein:

Wert	Bedeutung
CC_ENABLEHOOK	Aktiviert die Filterfunktion, die im Element *lpfnHook* angegeben ist.
CC_ENABLE-TEMPLATE	Veranlaßt das System, die Dialogfensterschablone zu benutzen, die durch das Element *hInstance* bezeichnet wird und auf die das Element *lpTemplate* zeigt.
CC_ENABLE-TEMPLATEHANDLE	Das Element *hInstance* bezeichnet einen Datenblock, der eine schon geladene Dialogfensterschablone enthält. Falls dieses Flag angegeben ist, ignoriert das System das Element *lpTemplateName*.
CC_FULLOPEN	Das Dialogfenster wird beim Anzeigen vollständig geöffnet. Es wird also auch der Teil angezeigt, der dem Benutzer die Erzeugung eigener Farben ermöglicht. Ohne dieses Flag muß der Benutzer den Schalter *Farben definieren* wählen, um diesen Teil anzuzeigen.
CC_PREVENT-FULLOPEN	Deaktiviert den Schalter *Farben definieren*. Der Benutzer kann keine selbstdefinierten Farben erzeugen.
CC_RGBINIT	Das Dialogfenster benutzt die Farbe, die im Element *rgbResult* angegeben ist, als Ausgangsfarbwahl.
CC_SHOWHELP	Das Dialogfenster zeigt den *Hilfe*-Schalter an. Wenn dieses Flag angegeben ist, darf das Element *hwndOwner* nicht *NULL* sein.

Diese Flags werden verwendet, wenn die Struktur initialisiert wird.

lCustData: Bestimmt die anwendungsdefinierten Daten, die das System der Filterfunktion übergibt, auf die das Element *lpfnHook* zeigt. Das System übergibt einen Zeiger auf die *CHOOSECOLOR*-Struktur im Parameter *lParam* der Botschaft *WM_INITDIALOG*. Dieser Zeiger kann zum Abrufen des Elements *lCustData* verwendet werden.

lpfnHook: Zeigt auf eine Filterfunktion, die Botschaften für das Dialogfenster bearbeitet. Um eine Filterfunktion zu aktivieren, muß eine Anwendung im Element *Flags* den Wert *CC_-ENABLEHOOK* angeben. Andernfalls ignoriert das System dieses Strukturelement. Die Filterfunktion muß Null zurückgeben, um eine nicht bearbeitete Botschaft an die Dialogfenster-

funktion in COMMDLG.DLL zurückzugeben. Die Filterfunktion muß einen Wert ungleich Null zurückgeben, um die Dialogfensterfunktion in COMMDLG.DLL davon abzuhalten, eine schon bearbeitete Botschaft nochmals zu bearbeiten. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

lpTemplateName: Zeigt auf einen nullterminierten String, der den Namen der Quelldatei für die Dialogfensterschablone bestimmt, die die Dialogfensterschablone in COMMDLG.DLL ersetzen soll. Eine Anwendung kann das Makro *MAKEINTRESOURCE* für numerierte Dialogfensterressourcen verwenden. Dieses Element wird nur verwendet, wenn das Element *Flags* das Flag *CC_ENABLETEMPLATE* enthält. Andernfalls wird es ignoriert. Das Element wird für die Initialisierung des Dialogfensters benötigt.

Hinweis

Einige Elemente dieser Struktur werden nur initialisiert, wenn das Dialogfenster erzeugt wird. Andere besitzen Initialisierungswerte, die beim Schließen des Dialogfensters aktualisiert werden. Ist im Abschnitt »Elemente« nicht angegeben, wann der Wert eines Elements zugewiesen wird, geschieht die Zuweisung beim Erzeugen des Dialogfensters.

Siehe auch

ChooseColor

CHOOSEFONT 3.1

Definition

```
#include <commdlg.h>

typedef struct tagCHOOSEFONT { /* cf                  */
 DWORD lStructSize;
 HWND hwndOwner;               /* Fenster-Handle der
                                  aufrufenden
                                  Anwendung           */
 HDC hDC;                      /* Gerätekontext-Handle
                                  oder NULL           */
 LOGFONT FAR* lpLogFont;       /* Zeiger auf eine
                                  LOGFONT-Struktur    */
 int iPointSize;               /* zehnfache Punktgröße
                                  der ausgewählten
                                  Schrift             */
 DWORD Flags;                  /* Aufzählungstyp-Flag */
 COLORREF rgbColors;           /* zurückgegebene
                                  Textfarbe           */
 LPARAM lCustData;             /* Daten, die an die
                                  Filterfunktion
                                  übergeben werden    */
 UINT (CALLBACK* lpfnHook)     /* Zeiger auf die      */
 (HWND, UINT, WPARAM, LPARAM); /* Filterfunktion      */
```

```
    LPCSTR lpTemplateName;      /* Name der anwendungs-
                                   definierten
                                   Schablone          */
    HINSTANCE hInstance;        /* Instanz-Handle der
                                   ausführbaren Datei,
                                   die die Schablone
                                   enthält            */
    LPSTR lpszStyle;            /* zeigt auf den Puffer
                                   der Auszeichnungs-
                                   beschreibung, der
                                   mindestens lf_Face-
                                   Size Byte groß ist  */
    UINT nFontType;             /* von EnumFontsCallback
                                   zurückgegebener Wert,
                                   zuzüglich der
                                   Bits fonttype_      */
    int nSizeMin;               /* kleinste erlaubte
                                   Punktgröße          */
    int nSizeMax;               /* größte erlaubte Punkt-
                                   größe, wenn cf_Limit-
                                   Size benutzt wird   */
} CHOOSEFONT;
```

Beschreibung Die *CHOOSEFONT*-Struktur enthält Informationen, die das System benutzt, um das systemdefinierte Dialogfenster *Schriftart* zu initialisieren. Nachdem der Benutzer den Schalter *OK* zum Schließen des Dialogfensters gewählt hat, liefert das System Informationen über die Auswahl des Benutzers in dieser Struktur zurück.

Elemente *lStructSize:* Bestimmt die Länge der Struktur in Byte. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

hwndOwner: Bezeichnet das Fenster, dem das Dialogfenster zugeordnet ist. Das Element kann jedes gültige Handle eines Fensters sein. Es sollte *NULL* sein, wenn das Dialogfenster keinen Besitzer hat.

Ist das Flag *CF_SHOWHELP* gesetzt, muß *hwndOwner* das Fenster bezeichnen, dem das Dialogfenster gehört. Die Fensterfunktion des besitzenden Fensters erhält eine Benachrichtigungsbotschaft, wenn der Benutzer den *Hilfe*-Schalter wählt. Der Bezeichner der Benachrichtigungsbotschaft ist der Wert, den die Funktion *RegisterWindowMessage* liefert, wenn *HELPMSGSTRING* als Argument übergeben wird. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

hDC: Bezeichnet den Gerätekontext oder den Informationskontext des Druckers, für den die Schriften im Dialogfenster aufgelistet werden sollen. Dieses Element wird nur benutzt,

wenn das Element *Flags* das Flag *CF_PRINTERFONTS* enthält. Andernfalls wird es ignoriert. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

lpLogFont: Zeigt auf eine *LOGFONT*-Struktur. Wenn eine Anwendung die Elemente dieser Struktur initialisiert und das Flag *CF_INITTOLOGFONTSTRUCT* setzt, ehe *ChooseFont* aufgerufen wird, initialisiert die Funktion *ChooseFont* das Dialogfenster mit der Schrifart, die die weitestgehende Übereinstimmung aufweist. Nachdem der Benutzer den Schalter *OK* gewählt hat, um das Dialogfenster zu schließen, setzt die Funktion *ChooseFont* die Elemente der *LOGFONT*-Struktur entsprechend der letzten Auswahl des Benutzers. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

iPointSize: Bestimmt die Größe der gewählten Schrift in Zehntelpunkten. Die Funktion *ChooseFont* setzt diesen Wert, nachdem der Benutzer den Schalter *OK* gewählt hat, um das Dialogfenster zu schließen.

Flags: Bestimmt die Initialisierungs-Flags des Dialogfensters. Dieses Element kann eine Kombination der folgenden Werte sein:

Wert	Bedeutung
CF_APPLY	Bestimmt, daß die Funktion *ChooseFont* den Schalter *Hinzufügen* aktiviert.
CF_ANSIONLY	Die Funktion *ChooseFont* beschränkt die Auswahl auf Schriften, die den Windows-Zeichensatz benutzen. Ist dieses Flag gesetzt, kann der Benutzer keine Schrift auswählen, die nur Symbole enthält.
CF_BOTH	Veranlaßt das Dialogfenster, die verfügbaren Drucker- und Bildschirmschriftarten aufzulisten. Das Element *hDC* bezeichnet entweder den Gerätekontext oder den Informationskontext, der dem Drucker zugeordnet ist.
CF_TTONLY	Die Funktion *ChooseFont* zeigt und erlaubt nur die Auswahl von TrueType-Schriften.
CF_EFFECTS	Die Funktion *ChooseFont* aktiviert Farbeffekte und die Effekte Unterstreichen und Durchstreichen. Ist dieses Flag gesetzt, können die Elemente *lfStrikeOut* und *lfUnderline* der *LOGFONT*-Struktur und das Element *rgbColors* der *CHOOSEFONT*-Struktur gesetzt werden, ehe *ChooseFont* aufgerufen wird.

Wert	Bedeutung
	Ist das Flag nicht gesetzt, kann die Funktion *ChooseFont* diese Elemente setzen, nachdem der Benutzer den Schalter *OK* gewählt hat, um das Dialogfenster zu schließen.
CF_ENABLEHOOK	Aktiviert die Filterfunktion, die im Element *lpfnHook* dieser Struktur angegeben ist.
CF_ENABLE-TEMPLATE	Das Element *hInstance* bezeichnet einen Datenblock, der die Dialogfensterschablone enthält, auf die *lpTemplate* zeigt.
CF_ENABLE-TEMPLATEHANDLE	Das Element *hInstance* bezeichnet einen Datenblock, der eine bereits geladene Dialogfensterschablone enthält. Wird dieses Flag angegeben, ignoriert das System das Element *lpTemplateName*.
CF_FIXEDPITCH-ONLY	Die Funktion *ChooseFont* wählt nur Schriften mit festen Zeichenabständen aus.
CF_FORCEFONT-EXIST	Die Funktion *ChooseFont* zeigt eine Fehlermeldung, wenn der Benutzer eine Schrift oder eine Schriftauszeichnung auszuwählen versucht, die nicht existiert.
CF_INITTOLOG-FONTSTRUCT	Die Funktion *ChooseFont* verwendet die *LOGFONT*-Struktur, auf die *lpLogFont* zeigt, um die Steuerelemente des Dialogfensters zu initialisieren.
CF_LIMITSIZE	Die Funktion *ChooseFont* wählt nur Schriftgrößen in dem Bereich, der durch die Elemente *nSizeMin* und *nSizeMax* gegeben ist.
CF_NOFACESEL	Im Kombinationsfenster für Schriften wird keine Auswahl angezeigt. Anwendungen verwenden dieses Flag, um die Auswahl mehrerer Schriften zu unterstützen. Dieses Flag wird für die Initialisierung des Dialogfensters benötigt und beim Schließen des Fensters gesetzt.
CF_NOOEMFONTS	Die Funktion *ChooseFont* läßt keine Auswahl von Vektorschriften zu. Dieses Flag hat denselben Wert wie *CF_NOVECTORFONTS*.
CF_NO-SIMULATIONS	Die Funktion *ChooseFont* läßt keine Simulation von Schriften durch die grafische Geräteschnittstelle (GDI) zu.

Wert	Bedeutung
CF_NOSIZESEL	Im Kombinationsfenster für die Schriftgröße wird keine Auswahl angezeigt. Anwendungen verwenden dieses Flag, um die Auswahl mehrerer Schriftgrößen zu unterstützen.
CF_NOSTYLESEL	Im Kombinationsfenster *Schriftstil* wird keine Auswahl angezeigt. Anwendungen verwenden dieses Flag, um die Auswahl mehrerer Auszeichnungen zu unterstützen.
CF_NOVECTOR-FONTS	Die Funktion *ChooseFont* erlaubt keine Auswahl von Vektorschriften. Dieses Flag hat denselben Wert wie *CF_NOOEMFONTS*.
CF_PRINTERFONTS	Das Dialogfenster listet nur die Schriften auf, die von dem Drucker unterstützt werden, der dem durch *hDC* bezeichneten Gerätekontext oder Informationskontext zugeordnet ist.
CF_SCALABLEONLY	Die Funktion *ChooseFont* läßt nur die Auswahl skalierbarer Schriften zu. Skalierbare Schriften sind Vektorschriften, einige Druckerschriften, TrueType-Schriften sowie Schriften, die mit anderen Algorithmen oder Techniken skaliert werden.
CF_SCREENFONTS	Das Dialogfenster listet nur die Bildschirmschriften auf, die vom System unterstützt werden.
CF_SHOWHELP	Veranlaßt das Dialogfenster, den *Hilfe*-Schalter anzuzeigen. Wenn diese Option angegeben wurde, darf *hwndOwner* nicht *NULL* sein.
CF_USESSTYLE	Das Element *lpszStyle* zeigt auf einen Puffer, der einen String mit einer Stilbeschreibung enthält. Diesen benutzt die Funktion *ChooseFont*, um das Fenster *Schriftstil* zu initialisieren. Nachdem der Benutzer den Schalter *OK* gewählt hat, um das Dialogfenster zu schließen, kopiert die Funktion *ChooseFont* die Stilbeschreibung für die Auswahl des Benutzers in diesen Puffer.
CF_WYSIWYG	Die Funktion *ChooseFont* läßt nur Schriften zu, die sowohl auf dem Drucker als auch am Bildschirm zur Verfügung stehen. Ist dieses Flag gesetzt, sollten auch die Flags *CF_BOTH* und *CF_SCALABLEONLY* gesetzt sein.

Diese Flags können gesetzt werden, wenn die Struktur initialisiert wird. Ausnahmen sind angegeben.

rgbColors: Ist das Flag *CF_EFFECTS* gesetzt, enthält dieses Element die Rot-, Grün- und Blauwerte (RGB), die die Funktion *ChooseFont* benutzen soll, um die Textfarbe zu setzen. Nachdem der Benutzer den Schalter *OK* gewählt hat, um das Dialogfenster zu schließen, enthält dieses Element die Farbwerte, die der Benutzer ausgewählt hat. Dieses Element wird für die Initialisierung des Dialogfensters benötigt und beim Schließen des Fensters gesetzt.

lCustData: Bestimmt anwendungsdefinierte Daten, die die Anwendung der Filterfunktion übergibt. Das System übergibt einen Zeiger auf die *CHOOSEFONT*-Datenstruktur im Parameter *lParam* der Botschaft *WM_INITDIALOG*. Das Element *lCustData* kann unter Benutzung dieses Zeigers abgerufen werden.

lpfnHook: Zeigt auf eine Filterfunktion, die die Botschaften für das Dialogfenster bearbeitet. Um die Filterfunktion zu aktivieren, muß eine Anwendung den Wert *CF_ENABLEHOOK* im Element *Flags* angeben. Andernfalls ignoriert das System dieses Strukturelement. Die Filterfunktion muß Null zurückgeben, um eine Botschaft an die Dialogfensterfunktion in COMMDLG.DLL weiterzugeben, die sie selber nicht bearbeitet hat. Die Filterfunktion muß einen Wert ungleich Null liefern, um die Dialogfensterfunktion in COMMDLG.DLL davon abzuhalten, eine bereits bearbeitete Botschaft nochmals zu bearbeiten. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

lpTemplateName: Zeigt auf einen nullterminierten String, der den Namen der Ressourcendatei der Dialogfensterschablone bestimmt, die die Dialogfensterschablone in COMMDLG.DLL ersetzen soll. Eine Anwendung kann das Makro *MAKEINTRESOURCE* für numerierte Dialogfensterressourcen verwenden. Dieses Element wird nur verwendet, wenn das Element *Flags* das Flag *CF_ENABLETEMPLATE* enthält. Andernfalls wird das Element ignoriert. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

hInstance: Bezeichnet einen Datenblock, der die Dialogfensterschablone enthält, die das Element *lpTemplateName* bestimmt. Dieses Element wird nur verwendet, wenn das Element *Flags* das Flag *CF_ENABLETEMPLATE* oder *CF_ENABLETEMPLATEHANDLE* enthält. Andernfalls wird das Element ignoriert. Dieses Element wird für die Initialisierung des Dialogfensters benötigt und beim Schließen des Fensters gesetzt.

lpszStyle: Zeigt auf einen Puffer, der einen String mit der Beschreibung der Schriftart enthält. Ist das Flag *CF_USESTYLE*

gesetzt, benutzt die Funktion *ChooseFont* die Daten in diesem Puffer, um das Fenster *Schriftstil* zu initialisieren. Wenn der Benutzer den Schalter *OK* gewählt hat, um das Dialogfenster zu schließen, kopiert die Funktion *ChooseFont* den String aus dem *Schriftstil*-Fenster in diesen Puffer.

Der Puffer, auf den *lpszStyle* zeigt muß mindestens *LF_FACESIZE* Byte lang sein.

Dieses Element wird für die Initialisierung des Dialogfensters benötigt und beim Schließen des Fensters gesetzt.

nFontType: Bestimmt den Typ der ausgewählten Schrift. Dieses Element kann einen oder mehrere der folgenden Werte enthalten:

Wert	Bedeutung
BOLD_FONTTYPE	Die Schrift ist fett. Dieser Wert bezieht sich nur auf TrueType-Schriften. Er entspricht dem Wert des Elements *ntmFlags* der *NEWTEXTMETRIC*-Struktur.
ITALIC_FONTTYPE	Die Schrift ist kursiv. Dieser Wert bezieht sich nur auf TrueType-Schriften. Er entspricht dem Wert des Elements *ntmFlags* der *NEWTEXTMETRIC*-Struktur.
PRINTER_FONTTYPE	Bestimmt, daß die Schrift eine Druckerschrift ist.
REGULAR_FONTTYPE	Die Schrift ist weder fett noch kursiv. Dieser Wert bezieht sich nur auf TrueType-Schriften. Er entspricht dem Wert des Elements *ntmFlags* der *NEWTEXTMETRIC*-Struktur.
SCREEN_FONTTYPE	Die Schrift ist eine Bildschirmschrift.
SIMULATED_FONTTYPE	Die Schrift wird durch die GDI simuliert. Dieses Flag ist nicht gesetzt, wenn das Flag *CF_NOSIMULATIONS* gesetzt ist.

nSizeMin: Bestimmt die minimale Punktgröße, die ein Benutzer wählen kann. Die Funktion *ChooseFont* erkennt dieses Element nur, wenn das Flag *CF_LIMITSIZE* gesetzt ist. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

nSizeMax: Bestimmt die maximale Punktgröße, die ein Benutzer wählen kann. Die Funktion *ChooseFont* erkennt dieses Element nur, wenn das Flag *CF_LIMITSIZE* gesetzt ist. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

Siehe auch *ChooseFont*

CLASSENTRY 3.1

Definition

```
#include <toolhelp.h>

typedef struct tagCLASSENTRY { /* ce */
   DWORD dwSize;
   HMODULE hInst;
   char szClassName[MAX_CLASSNAME + 1];
   WORD wNext;
} CLASSENTRY;
```

Beschreibung Die *CLASSENTRY*-Struktur enthält den Namen einer Windows-Klasse und einen **near**-Zeiger auf die nächste Klasse in der Liste.

Elemente *dwSize:* Bestimmt die Größe der *CLASSENTRY*-Struktur in Byte.

hInst: Bezeichnet das Instanz-Handle der Task, der die Klasse zugeordnet ist. Eine Anwendung benötigt dieses Handle, um *GetClassInfo* aufzurufen. Das Element *hInst* ist in Wirklichkeit ein Handle für ein Modul, da Windows-Klassen Modulen zugeordnet werden. Deshalb entspricht dieses *hInst* nicht dem Parameter *hInst*, der der Funktion *WinMain* der besitzenden Task übergeben wird.

szClassName: Bestimmt den nullterminierten String, der den Klassennamen enthält. Eine Anwendung benötigt diesen Namen, um *GetClassInfo* aufzurufen.

wNext: Bestimmt die nächste Klasse in der Liste. Dieses Element ist für den internen Gebrauch durch Windows reserviert.

Siehe auch *ClassFirst*, *ClassNext*

COMSTAT 3.1

Definition

```
typedef struct tagCOMSTAT { /* cmst                  */
   BYTE status;             /* Status der übertragung */
   UINT cbInQue;            /* Anzahl der Zeichen
                               in der Empfangs-
                               warteschlange          */
   UINT cbOutQue;           /* Anzahl der Zeichen
                               in der übertragungs-
                               warteschlange          */
}  COMSTAT;
```

Beschreibung Die *COMSTAT*-Struktur enthält Informationen über eine serielle Schnittstelle.

Elemente *status:* Bestimmt den Status der Übertragung. Dieses Element kann eines oder mehrere der folgenden Flags sein:

Flag	Bedeutung
CSTF_CTSHOLD	Bestimmt, ob die Übertragung darauf wartet, daß das CTS-Signal (clear-to-send) gesendet wird.
CSTF_DSRHOLD	Bestimmt, ob die Übertragung darauf wartet, daß das DSR-Signal (data-set-ready) gesendet wird.
CSTF_RLSDHOLD	Bestimmt, ob die Übertragung darauf wartet, daß das RLSD-Signal (receive-line-signal-detect) gesendet wird.
CSTF_XOFFHOLD	Bestimmt, ob die Übertragung wartet, wenn das Zeichen XOFF empfangen wurde.
CSTF_XOFFSENT	Bestimmt, ob die Übertragung wartet, wenn das Zeichen XOFF übertragen wurde. Die Übertragung muß nach dem Senden eines XOFF-Zeichens für Systeme gestoppt werden, die das folgende Zeichen unabhängig von seinem tatsächlichen Wert als XON interpretieren.
CSTF_EOF	Bestimmt, ob das Zeichen für Dateiende (EOF) empfangen worden ist.
CSTF_TXIM	Bestimmt, ob ein Zeichen darauf wartet, übertragen zu werden.

cbInQue: Bestimmt die Anzahl der Zeichen in der Empfangswarteschlange.

cbOutQue: Bestimmt die Anzahl der Zeichen in der Ausgabewarteschlange.

Siehe auch *GetCommError*

CONVCONTEXT 3.1

Definition

```
#include <ddeml.h>

typedef struct tagCONVCONTEXT {  /* cc                */
  UINT cb;              /* Größe der Struktur        */
  UINT wFlags;          /* nicht definiert           */
  UINT wCountryID;      /* verwendeter Ländercode für
                           Themen- und Element-Strings */
  int iCodePage;        /* verwendete Code-Seite für
                           Themen- und Element-Strings */
  DWORD dwLangID;       /* Sprachbezeichner für
                           Themen- und Element-Strings */
```

```
  DWORDdwSecurity;     /* privater Sicherheitscode    */
} CONVCONEXT;
```

Beschreibung Die *CONVCONTEXT*-Struktur enthält Informationen, die es Anwendungen ermöglichen, Daten in verschiedenen Sprachen gemeinsam zu benutzen.

Elemente *cb:* Bestimmt die Größe der *CONVCONTEXT*-Struktur in Byte.

wFlags: Bestimmt die Flags des Konversationskontexts. Zur Zeit sind keine Flags für dieses Element definiert.

wCountryFlags: Bestimmt die Ländercodebezeichner für die Strings mit den Themen- und Elementnamen.

iCodePage: Bestimmt die Code-Seite für Themennamen- und Elementnamen-Strings. Einsprachige Clients sollten dieses Element auf *CP_WINANSI* setzen. Anwendungen, die den OEM-Zeichensatz verwenden, sollten dieses Element auf den Wert setzen, den die Funktion *GetKBCodePage* zurückgibt.

dwLangID: Bestimmt den Sprachbezeichner für Themennamen- und Elementnamen-Strings.

dwSecurity: Bestimmt einen privaten (anwendungsdefinierten) Sicherheitscode.

Siehe auch *GetKBCodePage*

CONVINFO 3.1

Definition

```
#include <ddeml.h>

typedef struct tagCONVINFO { /* ci                    */
   DWORD cb;                 /* Größe der Struktur    */
   DWORD hUser;              /* anwendungsdefinierte
                                Daten                 */
   HCONV hConvPartner;       /* Handle für den
                                Konversationspartner  */
   HSZ hszSvcPartner;        /* Anwendungsname des
                                Partners              */
   HSZ hszServiceReq;        /* Servicename für die
                                Konversation          */
   HSZ hszTopic;             /* Themenname für die
                                Konversation          */
   HSZ hszItem;              /* Elementname für die
                                Konversation          */
   UINT wFmt;                /* übergabeformat        */
   UINT wType;               /* Flag für den
                                übertragungstyp       */
   UINT wStatus;             /* Status der aktuellen
                                Konversation          */
```

```
    UINT wConvst;           /* Konversationsstatus   */
    UINT wLastError;        /* letzter Transaktions-
                               fehler                */
    HCONVLIST hConvList;    /* Handle der
                               Konversationsliste    */
    CONVCONTEXT ConvCtxt;   /* Konversationskontext  */
} CONVINFO;
```

Beschreibung Die *CONVINFO*-Struktur enthält Informationen über eine DDE-Konversation.

Elemente *cb:* Bestimmt die Länge der Struktur in Byte.

hUser: Bezeichnet die anwendungsdefinierten Daten.

hConvPartner: Bezeichnet die Partneranwendung in der DDE-Konversation. Falls der Partner sich nicht mit Hilfe der Funktion *DdeInitialize* registriert hat, um Funktionen aus der DDE-Verwaltungsbibliothek (DDEEML) aufzurufen, wird dieses Element auf Null gesetzt.

Eine Anwendung sollte dieses Element keiner DDEML-Funktion außer *DdeQueryConvInfo* übergeben.

hszSvcPartner: Bezeichnet den Servicenamen der Partneranwendung.

hszServiceReq: Bezeichnet den Servicenamen der Server-Anwendung, mit der eine Verbindung angefordert wurde.

hszTopic: Bezeichnet den Namen des angeforderten Themas.

hszItem: Bezeichnet den Namen des angeforderten Elements. *hszItem* ist transaktionsspezifisch.

wFmt: Bestimmt das Format der Daten, die ausgetauscht werden. *wFmt* ist transaktionsspezifisch.

wType: Bestimmt den Typ der aktuellen Transaktion. Dieses Element ist transaktionsspezifisch und kann einen der folgenden Werte annehmen:

Wert	Bedeutung
XTYP_ADVDATA	Informiert einen Client, daß Advise-Daten von einem Server angekommen sind.
XTYP_ADVREQ	Fordert während einer Advise-Schleife von einem Server das Senden aktualisierter Daten an einen Client an. Diese Transaktion ergibt sich aus dem Aufruf der Funktion *DdePost-Advise* durch den Server.
XTYP_ADVSTART	Fordert von einem Server den Beginn einer Advise-Schleife mit einem Client an.

Wert	Bedeutung
XTYP_ADVSTOP	Benachrichtigt einen Server, daß eine Advise-Schleife endet.
XTYP_CONNECT	Fordert von einem Server das Einrichten einer Konversation mit einem Client an.
XTYP_CONNECT-CONFIRM	Benachrichtigt einen Server, daß eine Konversation mit einem Client eingerichtet worden ist.
XTYP_DISCONNECT	Benachrichtigt einen Server, daß eine Konversation mit einem Client beendet worden ist.
XTYP_ERROR	Benachrichtigt eine DDEML-Anwendung, daß ein kritischer Fehler aufgetreten ist. Die DDEML könnte nicht genügend Ressourcen besitzen, um fortzufahren.
XTYP_EXECUTE	Verlangt, daß ein Server einen Befehl ausführt, den er von einem Client empfangen hat.
XTYP_MONITOR	Benachrichtigt eine Anwendung, die als *APPCMD_MONITOR* registriert ist, daß DDE-Daten übertragen werden.
XTYP_POKE	Fordert von einem Server die Annahme nicht angeforderter Daten von einem Client an.
XTYP_REGISTER	Benachrichtigt andere DDEML-Anwendungen, daß ein Server einen Servicenamen registriert hat.
XTYP_REQUEST	Verlangt, daß ein Server Daten an einen Client sendet.
XTYP_UNREGISTER	Benachrichtigt andere DDEML-Anwendungen, daß ein Server die Registrierung eines Servicenamens aufgehoben hat.
XTYP_WILD-CONNECT	Verlangt, daß ein Server mehrere Konversationen mit demselben Client einrichtet.
XTYP_XACT_-COMPLETE	Benachrichtigt einen Client, daß eine asynchrone Transaktion vollendet worden ist.

wStatus: Bestimmt den Status der aktuellen Konversation. Dieses Element kann eine Kombination der folgenden Werte sein:

ST_ADVISE
ST_BLOCKED
ST_BLOCKNEXT

ST_CLIENT
ST_CONNECTED
ST_INLIST
ST_ISLOCAL
ST_ISSELF
ST_TERMINATED

wConvst: Bestimmt den Zustand der Konversation. Dieses Element kann einer der folgenden Werte sein:

XST_ADVACKRCVD
XST_ADVDATAACKRCVD
XST_ADVDATASENT
XST_ADVSENT
XST_CONNECTED
XST_DATARCVD
XST_EXECACKRCVD
XST_EXECSENT
XST_INCOMPLETE
XST_INIT1
XST_INIT2
XST_NULL
XST_POKEACKRCVD
XST_POKESENT
XST_REQSENT
XST_UNADVACKRCVD
XST_UNADVSENT

wLastError: Bestimmt den Fehlerwert der letzten Transaktion.

hConvList: Bezeichnet eine Konversationsliste, wenn das Handle der aktuellen Konversation in einer Konversationsliste eingetragen ist. Andernfalls ist dieses Element *NULL*.

ConvCtxt: Bestimmt den Kontext der Konversation.

Siehe auch *CONVCONTEXT*

CPLINFO 3.1

Definition

```
#include <cpl.h>

typedef struct tagCPLINFO {  /* cpli                  */
   int idIcon;               /* Bezeichner der
                                Symbolressource       */
   int idName;               /* Bezeichner der
                                String-Ressource für
                                den Anwendungsnamen   */
   int idInfo;               /* Bezeichner der String-
                                Ressource für die An-
                                wendungsbeschreibung  */
```

```
   LONG lData;              /* benutzerdefinierte
                               Daten                */
} CPLINFO;
```

Beschreibung Die *CPLINFO*-Struktur enthält Informationen über Ressourcen und einen benutzerdefinierten Wert für eine erweiterbare Anwendung der Systemsteuerung

Elemente *idIcon:* Bestimmt den Bezeichner einer Symbolressource für das Symbol der Anwendung. Dieses Symbol wird im Fenster der Systemsteuerung angezeigt.

idName: Bestimmt den Bezeichner einer String-Ressource für den Anwendungsnamen. Der Name ist ein kurzer String, der unter dem Symbol der Anwendung im Fenster der Systemsteuerung angezeigt wird. Der Name wird auch im Menü *Einstellungen* der Systemsteuerung angezeigt.

idInfo: Bestimmt den Bezeichner einer String-Ressource für die Anwendungsbeschreibung. Dieser String wird unten im Fenster der Systemsteuerung angezeigt, wenn das Symbol der Anwendung ausgewählt ist.

lData: Bestimmt benutzerdefinierte Daten für die Anwendung.

CTLINFO 3.1

Definition

```
#include <custcntl.h>

typedef struct tagCTLINFO {
  UINT wVersion;           /* Versionsnummer der
                              Steuerelementklasse     */
  UINT wCtlTypes;          /* Anzahl der Typen,
                              die von der Klasse
                              unterstützt werden      */
  char szClass[CTLCLASS];  /* Name der
                              Steuerelementklasse     */
  char szTitle[CTLTITLE];  /* Copyright-Informationen */
  char szReserved[10];     /* für zukünftige Nutzung
                              reserviert              */
  CTLTYPE  Type[CTLTYPES]; /* Informationen über die
                              unterstützten Typen     */
} CTLINFO;
```

Beschreibung Die *CTLINFO*-Struktur definiert den Klassennamen und die Versionsnummer für ein anwendungsdefiniertes Steuerelement. Die *CTLINFO*-Struktur enthält außerdem ein Array von *CTLTYPE*-Strukturen. Jede von ihnen listet gebräuchliche Kombinationen von Stilen auf (auch Varianten genannt), zusammen mit einer kurzen Beschreibung und Informationen über die vorgeschlagene Größe.

Elemente

wVersion: Bestimmt die Versionsnummer des Steuerelements. Obwohl man das Numerierungsschema mit einer Ziffer beginnen kann, verwenden die meisten Implementierungen die letzten beiden Ziffern für die Darstellung der Unterversionsnummern.

wCtlTypes: Bestimmt die Anzahl der Steuerelementtypen, die von dieser Klasse unterstützt werden. Dieser Wert sollte immer größer Null und kleiner oder gleich dem Wert *CTLTYPES* sein.

szClass: Bestimmt einen nullterminierten String, der den Namen der Steuerelementklasse enthält, der von der dynamischen Linkbibliothek (DLL) unterstützt wird. Der String sollte nicht länger als der Wert von *CTLCLASS* sein.

szTitle: Bestimmt einen nullterminierten String, der verschiedene Informationen über Copyright und Autoren enthält. Die Informationen beziehen sich auf die Steuerelementbibliothek. Der String sollte nicht länger als der Wert von *CTLTITLE* sein.

Type: Bestimmt ein Array von *CTLTYPE*-Strukturen, die Informationen über jeden Steuerelementtyp enthalten, der von dieser Klasse unterstützt wird. Das Array sollte nicht mehr Strukturen besitzen, als der Wert *CTLTYPES* angibt.

Hinweis

Eine Anwendung ruft die Informationsfunktion der entsprechenden Klasse auf, um grundlegende Informationen über die Steuerelementbibliothek abzurufen. Aufgrund der erhaltenen Informationen kann die Anwendung Instanzen eines Steuerelements erzeugen, indem sie einen der unterstützten Stile verwendet. Zum Beispiel ruft der Dialog-Editor diese Funktion auf, um die Steuerelementstile zu ermitteln, die eine Bibliothek anzeigen kann.

Der Rückgabewert der Klasseninformationsfunktion bezeichnet eine *CTLINFO*-Struktur, wenn die Funktion erfolgreich war. Diese Information wird der aufrufenden Anwendung zugeordnet, die sie explizit mit Hilfe der Funktion *GlobalFree* freigeben muß, wenn die Struktur nicht mehr benötigt wird.

Siehe auch

CTLSTYLE, CTLTYPE

CTLSTYLE 3.1

Definition

```
#include <custcntl.h>

typedef struct tagCTLSTYLE {
   UINT wX;                 /* x-Koordinate des
                               Ursprungs            */
   UINT wY;                 /* y-Koordinate des
                               Ursprungs            */
```

```
    UINT wCx;                /* Breite des
                                Steuerelements       */
    UINT wCy;                /* Höhe des
                                Steuerelements       */
    UINT wId;                /* Bezeichner des
                                Steuerelements       */
    DWORDdwStyle;            /* Stil-Flags für das
                                Steuerelement        */
    char szClass[CTLCLASS];  /* Name der
                                Steuerelementklasse  */
    char szTitle[CTLTITLE];  /* Text des
                                Steuerelements       */
} CTLSTYLE;
```

Beschreibung Die *CTLSTYLE*-Struktur bestimmt die Attribute des ausgewählten Steuerelements einschließlich der aktuellen Stil-Flags, der Position, der Abmessungen und des zugeordneten Texts.

Elemente *wX:* Bestimmt die x-Bildschirmkoordinate des Ursprungs des Steuerelements relativ zum Client-Bereich des übergeordneten Fensters.

wY: Bestimmt die y-Bildschirmkoordinate des Ursprungs des Steuerelements relativ zum Client-Bereich des übergeordneten Fensters.

wCx: Bestimmt die aktuelle Breite des Steuerelements in Bildschirmkoordinaten.

wCy: Bestimmt die aktuelle Höhe des Steuerelements in Bildschirmkoordinaten

wId: Bestimmt den Bezeichner des aktuellen Steuerelements. In den meisten Fällen sollte es Benutzern nicht erlaubt sein, diesen Wert zu verändern, da der Dialog-Editor ihn automatisch mit einer Header-Datei abstimmt.

dwStyle: Bestimmt den Stil des aktuellen Steuerelements. Das höherwertige *WORD* enthält die steuerelementspezifischen Flags, das niederwertige *WORD* die Windows-spezifischen. Der Benutzer kann diese Flags beliebig verändern, wenn die neuen Werte von der Steuerelementbibliothek unterstützt werden.

szClass: Bestimmt einen nullterminierten String, der den Namen der aktuellen Steuerelementklasse enthält. Der Benutzer darf dieses Element nicht ändern, es wird nur zu Informationszwecken bereitgestellt. Dieser String sollte nicht länger als der Wert von *CTLCLASS* sein.

szTitle: Bestimmt einen nullterminierten String, der den Text des Steuerelements enthält. Dieser Text wird normalerweise innerhalb des Steuerelements angezeigt. Er kann aber auch andere Informationen enthalten, die für das Steuerelement

benötigt werden. Dieser String sollte nicht länger als der Wert von *CTLTITLE* sein.

Hinweis

Eine Anwendung ruft die Stilfunktion der entsprechenden Klasse auf, um ein Dialogfenster anzuzeigen, in dem der Stil des ausgewählten Steuerelements bearbeitet werden kann. Wenn diese Funktion aufgerufen wird, sollte sie ein modales Dialogfenster anzeigen, in dem der Benutzer die Elemente von *CTLSTYLE* bearbeiten kann. Die Benutzeroberfläche des Dialogfensters sollte mit den vordefinierten Steuerelementen konsistent sein, die der Dialog-Editor unterstützt.

Siehe auch

CTLINFO, CTLTYPE

CTLTYPE 3.1

Definition

```
#include <custcntl.h>

typedef struct tagCTLTYPE {
   UINT wType;              /* reserviert, muß Null
                               sein                  */
   UINT wWidth;             /* vorgeschlagene Breite  */
   UINT wHeight;            /* vorgeschlagene Höhe    */
   DWORD dwStyle            /* voreingestellter Stil  */
   char szDescr[CTLDESCR];  /* Name des
                               Steuerelements        */
} CTLTYPE;
```

Beschreibung

Die *CTLTYPE*-Struktur enthält Informationen über ein Steuerelement in einer bestimmten Klasse. Die *CTLINFO*-Struktur enthält ein Array von *CTLTYPE*-Strukturen.

Elemente

wType: Reserviert, muß Null sein.

wWidth: Bestimmt die vorgeschlagene Breite des Steuerelements, wenn es mit dem Dialog-Editor erzeugt wird. Die Breite ist in Koordinaten des Ressourcencompilers angegeben.

wHeight: Bestimmt die vorgeschlagene Höhe des Steuerelements, wenn es mit dem Dialog-Editor erzeugt wird. Die Höhe ist in Koordinaten des Ressourcencompilers angegeben.

dwStyle: Bestimmt die anfänglichen Stil-Bits, die verwendet werden, um diesen Steuerelementtyp zu erzeugen. Dieser Wert enthält die steuerelementspezifischen Flags im höherwertigen und die Windows-spezifischen Flags im niederwertigen *WORD*.

szDescr: Definiert den Namen, der von anderen Programmierwerkzeugen verwendet wird, wenn sie sich auf diese spezielle Variante der zugrundeliegenden Steuerelementklasse beziehen. Der Dialog-Editor bezieht sich nicht auf diese Informa-

tion. Dieser String sollte nicht länger als der Wert von *CTLDESCR* sein.

Siehe auch *CTLINFO, CTLSTYLE*

DDEACK 2.x

Definition

```
#include <dde.h>
typedef struct tagDDEACK {  /* ddeack */
  WORD bAppReturnCode:8,
     reserved:6,
     fBusy:1,
     fAck:1;
} DDEACK;
```

Beschreibung Die *DDEACK*-Struktur enthält Status-Flags, die eine DDE-Anwendung ihren Partnern als Teil der Botschaft *WM_DDE_ACK* übergibt. Die Flags liefern Details über die Antwort der Anwendung auf eine *WM_DDE_ADVISE*-, *WM_DDE_DATA*-, *WM_DDE_EXECUTE*-, *WM_DDE_REQUEST*-, *WM_DDE_POKE*- oder *WM_DDE_UNADVISE*-Botschaft.

Elemente *bAppReturnCode:* Bestimmt einen anwendungsdefinierten Rückgabewert.

fBusy: Zeigt an, ob die Anwendung beschäftigt war und deshalb nicht auf die Botschaft des Partners antworten konnte. Ein Wert ungleich Null zeigt an, daß der Server beschäftigt war und nicht antworten konnte. Das Element *fBusy* ist nur definiert, wenn das Element *fAck* Null ist.

fAck: Zeigt an, ob die Anwendung die Botschaft vom Partner angenommen hat. Ein Wert ungleich Null zeigt an, daß der Server die Botschaft akzeptiert hat.

Siehe auch *WM_DDE_ACK, WM_DDE_ADVISE, WM_DDE_DATA, WM_DDE_EXECUTE, WM_DDE_POKE, WM_DDE_REQUEST, WM_DDE_UNADVISE*

DDEADVISE 2.x

Definition

```
#include <dde.h>
typedef struct tagDDEADVISE {  /* ddeadv */
  WORD reserved:14,
     dDeferUpd:1,
     fAckReq:1,
  short  cfFormat;
} DDEADVISE;
```

Beschreibung Die *DDEADVISE*-Struktur enthält Flags, die bestimmen, wie ein Server während einer Advise-Schleife Daten an einen

Client senden soll. Ein Client übergibt einem Server ein Handle für eine *DDEADVISE*-Struktur als Teil einer *WM_DDE_ADVISE*-Botschaft.

Elemente

fDeferUpd: Zeigt an, wann der Server aktualisierte Daten an den Client senden soll. Ein Wert ungleich Null weist den Server an, eine *WM_DDE_DATA*-Botschaft mit einem *NULL*-Daten-Handle zu senden, sobald sich das Datenelement ändert. Als Antwort kann der Client eine *WM_DDE_REQUEST*-Botschaft an den Server übergeben, um ein Handle für die aktualisierten Daten zu erhalten.

fAckReq: Zeigt an, ob der Server das Flag *fAckReq* in den *WM_DDE_DATA*-Botschaften setzen soll, die er an den Client übergibt. Ein Wert ungleich Null führt zum Setzen dieser Bits.

cfFormat: Bestimmt das bevorzugte Datenformat der Client-Anwendung. Das Format muß ein registriertes oder ein Standardzwischenablageformat sein. Die folgenden Standardzwischenablageformate können verwendet werden:

CF_BITMAP
CF_DCF_OEMTEXT
CF_DCF_PALETTE
CF_DCF_PENDATA
CF_DCF_SYLK
CF_DCF_TEXT
CF_METALFILEPICT
CF_OEMTEXT
CF_PALETTE
CF_PENDATA
CF_SYLK
CF_TEXT
CF_TIFF

Siehe auch

WM_DDE_ADVISE, WM_DDE_DATA, WM_DDE_UNADVISE

DDEDATA

2.x

Definition

```
#include <dde.h>

typedef struct tagDDEDATA {  /* ddeat */
  WORD unused:12,
     fResponse:1,
     fRelease:1,
     reserved:1,
     fAckReq:1;
  short cfFormat;
  Byte Value[1];
} DDEDATA;
```

Beschreibung Die *DDEDATA*-Struktur enthält die Daten und Informationen über die Daten, die als Teil der Botschaft *WM_DDE_DATA* gesendet werden.

Elemente *fResponse:* Zeigt an, ob eine Anwendung, die die Botschaft *WM_DDE_DATA* empfängt, den Empfang der Daten durch Senden der Botschaft *WM_DDE_ACK* bestätigen soll. Ein Wert ungleich Null zeigt an, daß die Anwendung die Bestätigung senden soll.

fRelease: Zeigt an, ob die Anwendung beim Empfang der Botschaft *WM_DDE_POKE* die Daten freigeben soll. Ein Wert ungleich Null bedeutet, daß die Daten freigegeben werden sollen.

fAckReq: Zeigt an, ob die Daten als Antwort auf eine *WM_DDE_REQUEST*- oder eine *WM_DDE_POKE*-Botschaft gesendet wurden. Ein Wert ungleich Null bedeutet, daß die Daten als Antwort auf eine *WM_DDE_REQUEST*-Botschaft gesendet wurden.

cfFormat: Bestimmt das Format der Daten. Das Format sollte ein registriertes oder ein Standardzwischenablageformat sein. Die folgenden Standardzwischenablageformate können verwendet werden:

CF_BITMAP
CF_DCF_OEMTEXT
CF_DCF_PALETTE
CF_DCF_PENDATA
CF_DCF_SYLK
CF_DCF_TEXT
CF_METALFILEPICT
CF_OEMTEXT
CF_PALETTE
CF_PENDATA
CF_SYLK
CF_TEXT
CF_TIFF

Siehe auch *WM_DDE_ACK*, *WM_DDE_ADVISE*, *WM_DDE_DATA*, *WM_DDE_POKE*, *WM_DDE_REQUEST*

DDEPOKE 2.x

Definition

```
#include <dde.h>

typedef struct tagDDEPOKE {     /* ddepok */
  WORD unused:13,
     fRelease:1,
     fReserved:2;
```

```
  short cfFormat;
  Byte Value[1];
} DDEPOKE;
```

Beschreibung Die *DDEPOKE*-Struktur enthält die Daten und Informationen über die Daten, die als Teil der Botschaft *WM_DDE_POKE* gesendet werden.

Elemente *fRelease:* Zeigt an, ob die Anwendung, die die Botschaft *WM_DDE_POKE* empfängt, die Daten freigeben soll. Ein Wert ungleich Null bedeutet, daß die Daten freigegeben werden sollen.

cfFormat: Bestimmt das Format der Daten. Das Format sollte ein registriertes oder ein Standardzwischenablageformat sein. Die folgenden Standardzwischenablageformate können verwendet werden:

CF_BITMAP
CF_DCF_OEMTEXT
CF_DCF_PALETTE
CF_DCF_PENDATA
CF_DCF_SYLK
CF_DCF_TEXT
CF_METALFILEPICT
CF_OEMTEXT
CF_PALETTE
CF_PENDATA
CF_SYLK
CF_TEXT
CF_TIFF

Value: Enthält die Daten. Die Größe des Array hängt vom Wert des Elements *cfFormat* ab.

Siehe auch *WM_DDE_POKE*

DEBUGHOOKINFO 3.1

Definition

```
typedef struct tagDEBUGHOOKINFO {
   HMODULE hModuleHook;
   LPARAM reserved;
   LPARAM lParam;
   WPARAM wParam;
   int code;
} DEBUGHOOKINFO;
```

Beschreibung Die *DEBUGHOOKINFO*-Struktur enthält Debug-Informationen.

Elemente *hModuleHook:* Bezeichnet das Modul, das die Filterfunktion enthält.

reserved: Wird nicht verwendet.

lParam: Bestimmt den Wert, der im Parameter *lParam* der Callback-Funktion *DebugProc* an den Filter übergeben werden soll.

wParam: Bestimmt den Wert, der im Parameter *wParam* der Callback-Funktion *DebugProc* an den Filter übergeben werden soll.

code: Bestimmt den Wert, der im Parameter *code* der Callback-Funktion *DebugProc* an den Filter übergeben werden soll.

Siehe auch *DebugProc, SetWindowsHook*

DEVNAMES 3.1

Definition

```
#include <commdlg.h>

typedef struct tagDEVNAMES {  /* dn */
   UINT wDriverOffset;
   UINT wDeviceOffset;
   UINT wOutputOffset;
   UINT wDefault;
   /* hier können optionale Daten vereinbart werden   */
} DEVNAMES;
```

Beschreibung Die *DEVNAMES*-Struktur enthält die Offsets von Strings, die den Treiber, den Namen und die Ausgabeschnittstelle eines Druckers bestimmen. Die Funktion *PrintDlg* benutzt diese Strings, um Steuerelemente im systemdefinierten Dialogfenster *Drucken* zu initialisieren. Wenn der Benutzer den Schalter *OK* wählt, um das Dialogfenster zu schließen, werden die Informationen über den gewählten Drucker in dieser Struktur zurückgegeben.

Elemente *wDriverOffset:* Bestimmt den Offset vom Anfang der Struktur bis zu einem nullterminierten String, der den DOS-Dateinamen des Gerätetreibers ohne Namenserweiterung enthält. Bei der Eingabe wird dieser String dazu benutzt, den anfänglich im Dialogfenster angezeigten Drucker festzulegen.

wDeviceOffset: Bestimmt den Offset vom Anfang der Struktur bis zu einem nullterminierten String, der den Namen des Geräts angibt. Dieser String darf einschließlich des Nullzeichens nicht länger als 32 Byte sein. Er muß außerdem identisch mit dem Element *dmDeviceName* der *DEVMODE*-Struktur sein.

wOutputOffset: Bestimmt den Offset vom Anfang der Struktur bis zu einem nullterminierten String, der den DOS-Gerätenamen der Ausgabeschnittstelle angibt.

wDefault: Gibt an, ob die Strings in der *DEVNAMES*-Struktur den voreingestellten Drucker bestimmen. So kann überprüft werden, ob sich der voreingestellte Drucker seit dem letzten Druck verändert hat. Dieses Element kann für die Initialisierung des Dialogfensters auf *DN_DEFAULTPRN* gesetzt werden. Falls dieses Flag gesetzt ist, werden die anderen Werte der *DEVNAMES*-Struktur mit dem aktuellen Standarddrucker verglichen.

Das Element *wDefault* wird nur verändert, wenn das Dialogfenster *Druckereinrichtung* angezeigt wurde und der Benutzer den Schalter *OK* gewählt hat, um es zu schließen. Wird der Standarddrucker gewählt, so wird das Flag *DN_DEFAULTPRN* gesetzt. Wird ein Drucker explizit ausgewählt, wird das Flag nicht gesetzt. Alle anderen Bits in diesem Element sind für den internen Gebrauch durch die Dialogfensterfunktion des Dialogfensters *Drucken* reserviert.

Siehe auch *PrintDlg*

DOCINFO 3.1

Definition

```
typedef struct tagDOCINFO {  /* di */
   int cbSize;
   LPCSTR lpszDocName;
   LPCSTR lpszOutput;
} DOCINFO;
```

Beschreibung Die *DOCINFO*-Struktur enthält die Eingabe- und Ausgabedateinamen, die von der Funktion *StartDoc* benutzt werden.

Elemente *cbSize:* Bestimmt die Größe der Struktur in Byte.

lpszDocName: Zeigt auf einen nullterminierten String, der den Namen des Dokuments bestimmt. Dieser String darf einschließlich des Nullzeichens nicht länger als 32 Zeichen sein.

lpszOutput: Zeigt auf einen nullterminierten String, der den Namen einer Ausgabedatei bestimmt. Dieser erlaubt die Umleitung eines Druckauftrags in eine Datei. Ist dieser Wert *NULL*, erfolgt die Ausgabe auf dem Gerät im angegebenen Gerätekontext.

Siehe auch *StartDoc*

DRIVERINFOSTRUCT 3.1

Definition

```
typedef struct tagDRIVERINFOSTRUCT {  /* drvinfst */
   UINT length;
   HDRVR hDriver;
   HINSTANCE hModule;
```

```
    char szAliasName[128];
} DRIVERINFOSTRUCT;
```

Beschreibung Die *DRIVERINFOSTRUCT*-Struktur enthält grundlegende Informationen über einen installierbaren Gerätetreiber.

Elemente *length:* Bestimmt die Größe der *DRIVERINFOSTRUCT*-Struktur.

hDriver: Bezeichnet eine Instanz des installierbaren Treibers.

hModule: Bezeichnet das Modul eines installierbaren Treibers.

szAliasName: Zeigt auf einen nullterminierten String, der den Treibernamen oder ein Alias angibt, unter dem der Treiber geladen wurde.

Siehe auch *GetDriverInfo*

DRVCONFIGINFO 3.1

Definition
```
typedef struct tagDRVCONFIGINFO {
    DWORD dwDCISize;
    LPCSTR lpszDCISectionName;
    LPCSTR lpszDCIAliasName;
} DRVCONFIGINFO;
```

Beschreibung Die *DRVCONFIGINFO*-Struktur enthält Informationen über die Einträge für einen installierbaren Gerätetreiber in der Datei SYSTEM.INI. Diese Struktur wird im Parameter *lParam* der Botschaften für installierbare Treiber (*DRV_CONFIGURE* und *DRV_INSTALL*) übergeben.

Elemente *dwDCISizeName:* Bestimmt die Größe der *DRVCONFIGINFO*-Struktur.

lpszDCISectionName: Zeigt auf einen nullterminierten String, der den Abschnittsnamen in der Datei *SYSTEM.INI* bestimmt, unter dem die Treiberinformationen aufgezeichnet sind.

lpszDCIAliasName: Zeigt auf einen nullterminierten String, der den Treibernamen oder den Alias bestimmt, unter dem der Treiber geladen wurde.

Siehe auch *DRV_GONFIGURE, DRV_INSTALL*

EVENTMSG 2.x

Definition
```
typedef struct tagEVENTMSG {  /* em */
    UINT message;
    UINT paramL;
    UINT paramH;
    UINT time;
} EVENTMSG;
```

Beschreibung Die *EVENTMSG*-Struktur enthält Informationen über die Windows-Anwendungswarteschlange. Diese Struktur wird verwendet, um Botschaftsinformationen für die Callback-Funktion *JournalPlaybackProc* zu speichern.

Elemente *message:* Bestimmt die Botschaftsnummer.

paramL: Bestimmt zusätzliche Informationen über die Botschaft. Die genaue Bedeutung hängt vom Wert des Elements *message* ab.

paramH: Bestimmt zusätzliche Informationen über die Botschaft. Die genaue Bedeutung hängt vom Wert des Elements *message* ab.

time: Bestimmt die Zeit, zu der die Botschaft übergeben wurde.

Siehe auch *JournalPlaybackProc, SetWindowsHook*

FINDREPLACE 3.1

Definition

```
#include <commdlg.h>

typedef struct tagFINDREPLACE {  /* fr                */
   DWORD lStructSize;      /* Größe der Struktur     */
   HWND hwndOwner;         /* Handle des besitzenden
                              Fensters               */
   HINSTANCE hInstance;    /* Instanz-Handle der aus-
                              führbaren Datei mit
                              der selbstdefinierten
                              Dialogfensterschablone */
   DWORD Flags;            /* FR_-Flags              */
   LPSTR lpstrFindWhat;    /* Zeiger auf den
                              Such-String            */
   LPSTR lpstrReplaceWith; /* Zeiger auf den
                              Ersatz-String          */
   UINT wFindWhatLen;      /* Größe des Puffers
                              für den Such-String    */
   UINT wReplaceWithLen;   /* Größe des Puffers für
                              den Ersatz-String      */
   LPARAM lCustData;       /* anwendungsdefinierte
                              Daten, die an die
                              Filterfunktion über-
                              geben werden           */
   UINT (CALLBACK* lpfnHook)
        (HWND, UINT, WPARAM, LPARAM);
   LPCSTR lpTemplateName;
} FINDREPLACE;
```

Beschreibung Die *FINDREPLACE*-Struktur enthält Informationen, die das System benutzt, um ein systemdefiniertes Dialogfenster *Suchen*

oder *Ersetzen* zu initialisieren. Nachdem der Benutzer den Schalter *OK* gewählt hat, um das Dialogfenster zu schließen, liefert das System Informationen über die Auswahl des Benutzers in dieser Struktur zurück.

Elemente

lStructSize: Bestimmt die Länge der Struktur in Byte. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

hwndOwner: Bezeichnet das Fenster, dem das Dialogfenster zugeordnet ist. Dieses Element kann jedes gültige Fenster-Handle, darf aber nicht *NULL* sein.

Ist das Flag *FR_SHOWHELP* gesetzt, muß *hwndOwner* das Fenster bezeichnen, dem das Dialogfenster zugeordnet ist. Die Fensterfunktion des besitzenden Fensters erhält eine Benachrichtigungsbotschaft, wenn der Benutzer den Schalter *Hilfe* wählt. Der Bezeichner dieser Benachrichtigungsbotschaft ist der Wert, den die Funktion *RegisterWindowMessage* zurückgibt, wenn ihr *HELPMSGSTRING* als Argument übergeben wird.

Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

hInstance: Bezeichnet einen Datenblock, der eine Dialogfensterschablone enthält, die das Element *lpTemplateName* bestimmt. Dieses Element wird nur verwendet, wenn das Element *Flags* die Flags *FR_ENABLETEMPLATE* oder *FR_ENABLETEMPLATEHANDLE* enthält. Andernfalls wird dieses Element ignoriert. Das Element wird für die Initialisierung des Dialogfensters benötigt.

Flags: Enthält die Initialisierungs-Flags des Dialogfensters. Dieses Element kann eine Kombination der folgenden Werte sein:

Wert	Bedeutung
FR_DIALOGTERM	Zeigt an, daß das Dialogfenster geschlossen wird. Das Fenster-Handle, das die Funktionen *FindText* oder *ReplaceText* zurückgegeben haben, ist nicht mehr gültig, nachdem dieses Bit gesetzt wurde. Dieses Flag wird vom System gesetzt.
FR_DOWN	Setzt die Richtung für eine Suche im Dokument. Ist das Flag gesetzt, verläuft eine Suche im Text nach unten. Ist das Flag gelöscht, verläuft die Suche nach oben. Anfangs bestimmt das Flag den Zustand der Schalter *Aufwärts* und *Abwärts*. Nachdem der Benutzer den Schalter *OK* gewählt hat, um das Dialogfenster zu schließen, bestimmt dieses Flag die Auswahl des Benutzers.

Wert	Bedeutung
FR_ENABLEHOOK	Aktiviert die Filterfunktion, die das Element *lpfnHook* dieser Struktur bezeichnet. Dieses Flag kann für die Initialisierung des Dialogfensters gesetzt werden.
FR_ENABLE-TEMPLATE	Veranlaßt das System die Dialogfensterschablone zu benutzen, die die Elemente *hInstance* und *hTemplateName* bezeichnen, um das Dialogfenster anzuzeigen. Dieses Flag wird nur benutzt, um das Dialogfenster zu initialisieren.
FR_ENABLE-TEMPLATEHANDLE	Zeigt an, daß das Element *hInstance* einen Datenblock bezeichnet, der eine bereits geladene Dialogfensterschablone enthält. Das System übergeht das Element *lpTemplate-Name*, wenn dieses Flag angegeben ist. Dieses Flag kann für die Initialisierung des Dialogfensters gesetzt werden.
FR_FINDNEXT	Zeigt an, daß die Anwendung nach dem nächsten Vorkommen des String suchen soll, den das Element *lpstrFindWhat* bestimmt. Dieses Flag wird vom System gesetzt.
FR_HIDEMATCH-CASE	Verbirgt und deaktiviert das Auswahlfeld *Groß-/Kleinschreibung*. Dieses Flag kann für die Initialisierung des Dialogfensters gesetzt werden.
FR_HIDEWHOLE-WORD	Verbirgt und deaktiviert das Auswahlfeld *Als Wort*. Dieses Flag kann für die Initialisierung des Dialogfensters gesetzt werden.
FR_HIDEUPDOWN	Verbirgt die Schaltfelder *Aufwärts* und *Abwärts*, die die Suchrichtung im Dokument festlegen. Dieses Flag kann für die Initialisierung des Dialogfensters gesetzt werden.
FR_MATCHCASE	Bestimmt, daß die Suche Groß- und Kleinschreibung berücksichtigen soll. Dieses Flag wird beim Erzeugen des Dialogfensters gesetzt und kann vom System als Reaktion auf Benutzereingaben verändert werden.
FR_NOMATCHCASE	Deaktiviert das Auswahlfeld *Groß-/Kleinschreibung*. Dieses Flag wird nur zum Initialisieren des Dialogfensters verwendet.
FR_NOUPDOWN	Deaktiviert die Schaltfelder *Aufwärts* und *Abwärts*. Dieses Flag wird nur zum Initialisieren des Dialogfensters verwendet.

Wert	Bedeutung
FR_NOWHOLE-WORD	Deaktiviert das Auswahlfeld *Als Wort*. Dieses Flag wird nur zum Initialisieren des Dialogfensters verwendet.
FR_REPLACE	Die Anwendung sollte das aktuelle Vorkommen des durch *lpstrFindWhat* bezeichneten String durch den String im Element *lpstrReplaceWith* ersetzen. Dieses Flag wird vom System gesetzt.
FR_REPLACEALL	Die Anwendung sollte alle Vorkommen des String im Element *lpstrFindWhat* durch den String in *lpstrReplaceWith* ersetzen. Dieses Flag wird vom System gesetzt.
FR_SHOWHELP	Veranlaßt das Dialogfenster, den *Hilfe*-Schalter anzuzeigen. Ist das Flag angegeben, darf *hwndOwner* nicht *NULL* sein. Dieses Flag kann für die Initialisierung des Dialogfensters gesetzt werden.
FR_WHOLEWORD	Markiert das Auswahlfeld *Als Wort*. Es werden nur ganze Wörter gefunden, die dem Such-String entsprechen. Das Flag wird beim Initialisieren des Dialogfensters gesetzt und vom System als Reaktion auf Benutzereingaben geändert.

lpstrFindWhat: Bestimmt den String, nach dem gesucht werden soll. Wenn beim Erzeugen des Dialogfensters bereits ein String bestimmt ist, wird das Editierfeld *Suchen nach* mit diesem String initialisiert. Falls das Flag *FR_FINDNEXT* beim Erzeugen des Dialogfensters gesetzt ist, sollte die Anwendung nach einem Vorkommen des String suchen. Dabei werden die Flags *FR_DOWN*, *FR_WHOLEWORD* und *FR_MATCHCASE* benutzt, um Richtung und Art der Suche genauer zu bestimmen. Die Anwendung muß einen Puffer für den String reservieren. Er sollte mindestens 80 Byte groß sein. Dieses Element wird beim Erzeugen des Dialogfensters gesetzt und kann entsprechend der Benutzereingaben vom System verändert werden.

lpstrReplaceWith: Bestimmt den Ersatz-String für Ersetzungsoperationen. Die Funktion *FindText* ignoriert dieses Element. Die Funktion *ReplaceText* verwendet diesen String, um das Editierfeld *Ersetzen durch* zu initialisieren. Dieses Element wird beim Erzeugen des Dialogfensters gesetzt und kann entsprechend der Benutzereingaben vom System verändert werden.

wFindWhatLen: Bestimmt die Länge des Puffers in Byte, auf den das Element *lpstrFindWhat* zeigt. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

wReplaceWithLen: Bestimmt die Länge des Puffers in Byte, auf den das Element *lpstrReplaceWith* zeigt. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

lCustData: Bestimmt die anwendungsdefinierten Daten, die das System der Filterfunktion übergibt, die das Element *lpfnHook* bezeichnet. Das System übergibt im Parameter *lParam* der Botschaft *WM_INITDIALOG* einen Zeiger auf eine *CHOOSECOLOR*-Struktur. Dieser Zeiger kann benutzt werden, um das Element *lCustData* abzurufen.

lpfnHook: Zeigt auf eine Filterfunktion, die Botschaften für das Dialogfenster bearbeitet. Um die Filterfunktion zu aktivieren, muß eine Anwendung das Flag *FR_ENABLEHOOK* im Element *Flags* angeben. Andernfalls ignoriert das System dieses Strukturelement. Die Filterfunktion muß den Wert Null zurückgeben, um eine nicht bearbeitete Botschaft an die Dialogfensterfunktion in der Datei COMMDLG.DLL zu übergeben. Die Filterfunktion muß einen Wert ungleich Null zurückgeben, um die nochmalige Bearbeitung der Botschaft durch die Dialogfensterfunktion zu verhindern. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

lpTemplateName: Zeigt auf einen nullterminierten String, der den Namen der Ressourcendatei für die Dialogfensterschablone bestimmt, die die Dialogfensterschablone in COMMDLG.DLL ersetzen soll. Eine Anwendung kann das Makro *MAKEINTRESOURCE* auf numerierte Dialogfensterressourcen anwenden. Dieses Element wird nur verwendet, wenn das Element *Flags* das Flag *FR_ENABLETEMPLATE* enthält, andernfalls wird es ignoriert. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

Hinweis

Einige Elemente dieser Struktur werden nur gefüllt, wenn das Dialogfenster erzeugt, andere wenn das Dialogfenster vom Benutzer geschlossen wird. Einige Elemente besitzen einen Initialisierungswert, der beim Schließen des Fensters durch den Benutzer verändert wird. Wenn in der Beschreibung der Elemente nicht angegeben ist, wie der Wert eines Elements zugewiesen wird, findet die Zuweisung beim Erzeugen des Dialogfensters statt.

Siehe auch

FindText, ReplaceText

FIXED 3.1

Definition

```
typedef struct tagFIXED { /* fx */
   UINT fract;
   int  value;
} FIXED;
```

Beschreibung Die *FIXED*-Struktur enthält den ganzzahligen und den gebrochenen Anteil einer reellen Festkommazahl.

Elemente *fract:* Bestimmt den gebrochenen Anteil der Zahl.

value: Bestimmt den ganzzahligen Anteil der Zahl.

Hinweis Die *FIXED*-Struktur wird verwendet, um die Elemente der Strukturen *MAT2* und *POINTFX* zu beschreiben.

Siehe auch *GetGlyphOutline*

FMS_GETDRIVEINFO

Definition

```
#include <wfext.h>

typedef struct tagFMS_GETDRIVEINFO { /* fmsgdi        */
   DWORD dwTotalSpace;
   DWORD dwFreeSpace;
   char szPath[260];    /* aktuelles Verzeichnis      */
   char szVolume[14];   /* Datenträgerbezeichnung     */
   char szShare[128];   /* falls Netzlaufwerk         */
} FMS_GETDRIVEINFO, FAR *LPFMS_GETDRIVEINFO;
```

Beschreibung Die *FMS_GETDRIVEINFO*-Struktur enthält Informationen über das Laufwerk, das im momentan aktiven Fenster des Datei-Managers ausgewählt ist.

Elemente *dwTotalSpace:* Gibt den gesamten Speicherplatz auf dem dem Laufwerk zugeordneten Datenträger in Byte an.

FreeSpace: Gibt den gesamten freien Speicherplatz auf dem dem Laufwerk zugeordneten Datenträger in Byte an.

szPath: Gibt einen nullterminierten String an, der den Pfad des aktuellen Verzeichnisses enthält.

szVolume: Gibt einen nullterminierten String an, der die Bezeichnung des dem Laufwerk zugeordneten Datenträgers enthält.

szShare: Gibt einen nullterminierten String an, der den Namen des Netzknotens enthält (wenn über ein Netzwerk auf das Laufwerk zugegriffen wird).

Siehe auch *FMExtensionProc*, *FM_GETDRIVEINFO*

FMS_GETFILESEL

Definition

```
#include <wfext.h>

typedef struct tagFMS_GETFILESEL {    /* fmsgfs       */
   UINT wTime;
   UINT wDate;
   DWORD dwSize;
   BYTE bAttr;
   char szName[260];                  /* vollständiger
                                         Pfadname     */
} FMS_GETFILESEL;
```

Beschreibung Die *FMS_GETFILESEL*-Struktur enthält Informationen über eine ausgewählte Datei im Verzeichnisfenster oder im Fenster *Suchergebnis* des Datei-Managers.

Elemente *wTime:* Gibt die Erstellungszeit der Datei an.

wDate: Gibt das Erstellungsdatum der Datei an.

dwSize: Gibt die Größe der Datei in Byte an.

bAttr: Gibt die Attribute der Datei an.

szName: Gibt einen nullterminierten OEM-String an, der den vollständigen Pfadnamen der ausgewählten Datei enthält. Bevor eine Erweiterungs-DLL diesen String anzeigt, sollte sie die Funktion *OemToAnsi* verwenden, um den String in einen Windows-ANSI-String zu konvertieren. Wenn ein String an das MS-DOS-Dateisystem übergeben werden soll, sollte er nicht konvertiert werden.

Siehe auch *FMExtensionProc*

FMS_LOAD

Definition

```
#include <wfext.h>

typedef struct tagFMS_LOAD {          /* fmsld          */
   DWORD dwSize;                      /* für Versions-
                                         überprüfung    */
   char szMenuName[MENU_TEXT_LEN];    /* Ausgabe        */
   HMENU hMenu;                       /* Ausgabe        */
   UINT wMenuDelta;                   /* Eingabe        */
} FMS_LOAD;
```

Beschreibung Die *FMS_LOAD*-Struktur enthält Informationen, durch die dem Datei-Manager ein benutzerdefiniertes Menü hinzugefügt werden kann, das von einer Erweiterungs-DLL des Datei-Managers bereitgestellt wird. Die Struktur gibt auch einen Deltawert an, den die Erweiterungsbibliothek zur Anpassung des

anwendungsdefinierten Menüs verwenden kann, wenn der Datei-Manager das Menü geladen hat.

Elemente *dwSize:* Gibt die Länge der Struktur in Byte an.

szMenuName: Enthält einen nullterminierten String für ein Menüelement, das im Hauptmenü des Datei-Managers erscheinen soll.

hMenu: Bezeichnet das Popup-Menü, das dem Hauptmenü des Datei-Managers hinzugefügt wird.

wMenuDelta: Legt den Deltawert des Menüelements fest. Um Konflikte mit seinen eigenen Menüelementen zu vermeiden, numeriert der Datei-Manager die Bezeichner der Menüelemente in dem durch *hMenu* bezeichneten Popup-Menü um, indem er diesen Deltawert zu jedem Bezeichner addiert. Wenn eine Erweiterungs-DLL ein Menüelement ändern muß, kann sie auf dieses zugreifen, indem sie den Deltawert zum Bezeichner des Menüelements addiert.

Der Wert dieses Elements kann sich von Arbeitssitzung zu Arbeitssitzung ändern.

Siehe auch *FMExtensionProc*

GLOBALENTRY 3.1

Definition

```
#include <toolhelp.h>

typedef struct tagGLOBALENTRY { /* ge */
   DWORD dwSize;
   DWORD dwAddress;
   DWORD dwBlockSize;
   HGLOBAL hBlock;
   WORD wcLock;
   WORD wcPageLock;
   WORD wFlags;
   BOOL wHeapPresent;
   HGLOBAL hOwner;
   WORD wType;
   WORD wData;
   DWORD dwNext;
   DWORD dwNextAlt;
} GLOBALENTRY;
```

Beschreibung Die *GLOBALENTRY*-Struktur enthält Informationen über ein Speicherobjekt auf dem globalen Heap.

Elemente *dwSize:* Gibt die Größe der *GLOBALENTRY*-Struktur in Byte an.

dwAddress: Gibt die lineare Adresse des globalen Speicherobjekts an.

dwBlockSize: Gibt die Größe des globalen Speicherobjekts in Byte an.

hBlock: Bezeichnet das globale Speicherobjekt.

wcLock: Bestimmt den Sperrenzähler. Wenn dieser Wert Null ist, ist das globale Speicherobjekt nicht gesperrt.

wcPageLock: Bestimmt den Seitensperrenzähler. Wenn dieser Wert Null ist, ist die Speicherseite nicht gesperrt.

wFlags: Gibt Zusatzinformationen über das Speicherobjekt an. Dieses Element kann den Wert *GF_PDB_OWNER* annehmen. In diesem Fall ist der Prozeßdatenblock (PDB) der Task der Besitzer des Speicherobjekts.

wHeapPresent: Zeigt an, ob ein lokaler Heap innerhalb des globalen Speicherobjekts existiert.

hOwner: Bezeichnet den Besitzer des globalen Speicherobjekts.

wType: Bezeichnet den Speichertyp des Objekts. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
GT_UNKNOWN	Der Speichertyp ist nicht bekannt.
GT_DGROUP	Das Objekt enthält das Standarddaten- und das Stack-Segment.
GT_DATA	Das Objekt enthält Programmdaten. Es kann außerdem Stack-Daten und Daten des lokalen Heap enthalten.
GT_CODE	Das Objekt enthält Programmcode. Wird *GT_CODE* angegeben, enthält das Element *wData* die Segmentnummer für den Code.
GT_TASK	Das Objekt enthält die Task-Datenbank.
GT_RESOURCE	Das Objekt enthält den in *wData* angegebenen Ressourcentyp.
GT_MODULE	Das Objekt enthält die Modul-Datenbank.
GT_FREE	Das Objekt gehört zum freien Speicherbereich.
GT_INTERNAL	Das Objekt ist für die interne Verwendung durch Windows reserviert.
GT_SENTINEL	Das Objekt ist entweder das erste oder das letzte Objekt auf dem globalen Heap.
GT_BURGERMASTER	Das Objekt enthält eine Tabelle, die Selektoren auf Arena-Handles abbildet.

wData: Wenn das Element *wType* nicht den Wert *GT_CODE* oder *GT_RESOURCE* hat, ist *wData* Null. Wenn *wType* den Wert *GT_CODE*, *GT_DATA* oder *GT_DGROUP* hat, enthält *wData* die Segementnummer des Codes. Wenn *wType* den Wert *GT_-RESOURCE* hat, gibt *wData* den Ressourcentyp an. Möglich sind folgende Werte:

Wert	Bedeutung
GD_ACCELERATORS	Das Objekt enthält Daten aus der Tastenkürzeltabelle.
GD_BITMAP	Das Objekt enthält Daten, die ein Bitmap beschreiben. Dies schließt die Farbtabelle und die Bits des Bitmap ein.
GD_CURSOR	Das Objekt enthält Daten, die eine Gruppe von Mauszeigern beschreiben. Dies schließt die Höhe, die Breite, die Farbzähler, die Bitzähler und die ordinalen Bezeichner der Mauszeiger ein.
GD_CURSOR-COMPONENT	Das Objekt enthält Daten, die einen einzelnen Mauszeiger beschreiben. Dies schließt Bitmap-Bits und Bitmasken für den Mauszeiger ein.
GD_DIALOG	Das Objekt enthält Daten, die Steuerelemente in einem Dialogfenster beschreiben.
GD_ERRTABLE	Das Objekt enthält Daten aus der Fehlertabelle.
GD_FONT	Das Objekt enthält Daten, die eine einzelne Schrift beschreiben. Diese Daten sind mit den Daten in einer Windows-Schrift-Datei (.FNT) identisch.
GD_FONTDIR	Das Objekt enthält Daten, die eine Gruppe von Schriften beschreiben. Dies schließt die Anzahl der Schriftarten in der Ressource und eine metrische Tabelle für jede dieser Schriften ein.
GD_ICON	Das Objekt enthält Daten, die eine Gruppe von Symbolen beschreiben. Dies schließt die Höhe, die Breite, die Farbzähler, die Bitzähler und die ordinalen Bezeichner der Symbole ein.
GD_ICON-COMPONENT	Das Objekt enthält Daten, die ein einzelnes Symbol beschreiben. Dies schließt Bitmap-Bits und Bitmaps für das Symbol ein.

Wert	Bedeutung
GD_MENU	Das Objekt enthält Daten für normale und für Popup-Menüelemente.
GD_NAMETABLE	Das Objekt enthält Daten aus der Namenstabelle.
GD_RCDATA	Das Objekt enthält Daten aus einer benutzerdefinierten Ressource.
GD_STRING	Das Objekt enthält Daten aus der String-Tabelle.
GD_USERDEFINED	Die Ressource hat einen unbekannten Ressourcenbezeichner oder ist ein anwendungsspezifisch benannter Typ.

dwNext: Reserviert für interne Verwendung durch Windows.

dwNextAlt: Reserviert für interne Verwendung durch Windows.

Siehe auch *GlobalEntryHandle*, *GlobalEntryModule*, *GlobalFirst*, *GlobalNext*, *GLOBALINFO*

GLOBALINFO 3.1

Definition

```
#include <toolhelp.h>

typedef struct tagGLOBALINFO { /* gi */
   DWORD dwSize;
   WORD wcItems;
   WORD wcItemsFree;
   WORD wcItemsLRU;
} GLOBALINFO;
```

Beschreibung Die *GLOBALINFO*-Struktur enthält Informationen über den globalen Heap.

Elemente *dwSize:* Gibt die Größe der *GLOBALINFO*-Struktur in Byte an.

wcItems: Gibt die Gesamtzahl der Elemente auf dem globalen Heap an.

wcItemsFree: Gibt die Anzahl der freien Elemente auf dem globalen Heap an.

wcItemsLRU: Bestimmt die Anzahl der am längsten nicht benutzten Elemente (LRU =least recently used) auf dem globalen Heap.

Siehe auch *GlobalInfo*, *GLOBALENTRY*

GLYPHMETRICS 3.1

Definition

```
typedef struct tagGLYPHMETRICS { /* gm */
   UINT gmBlackBoxX;
   UINT gmBlackBoxY;
   POINT gmptGlyphOrigin;
   int gmCellIncX;
   int gmCellIncY;
} GLYPHMETRICS;
```

Beschreibung Die *GLYPHMETRICS*-Struktur enthält Informationen über die Plazierung und Ausrichtung eines Zeichenkörpers in einer Zeichenzelle.

Elemente *gmBlackBoxX:* Gibt die Breite des kleinsten Rechtecks an, das den Zeichenkörper vollständig umschließt.

gmBlackBoxY: Gibt die Höhe des kleinsten Rechtecks an, das den Zeichenkörper vollständig umschließt.

gmptGlyphOrigin: Gibt die x- und y-Koordinaten der oberen linken Ecke des Rechtecks an, das den Zeichenkörper vollständig umschließt.

gmCellIncX: Gibt den horizontalen Abstand des Ursprungs der aktuellen Zeichenzelle zum Ursprung der nächsten Zeichenzelle an.

gmCellIncY: Gibt den vertikalen Abstand des Ursprungs der aktuellen Zeichenzelle zum Ursprung der nächsten Zeichenzelle an.

Hinweis Die Werte in der *GLYPHMETRICS*-Struktur werden in virtuellen Einheiten angegeben.

Siehe auch *GetGlyphOutline*

HARDWAREHOOKSTRUCT 3.1

Definition

```
typedef struct tagHARDWAREHOOKSTRUCT { /* hhs */
   HWND hWnd;
   UINT wMessage;
   WPARAM wParam;
   LPARAM lParam;
} HARDWAREHOOKSTRUCT;
```

Beschreibung Die *HARDWAREHOOKSTRUCT*-Struktur enthält Informationen über eine Hardware-Botschaft, die sich in der Botschaftswarteschlange des Systems befindet.

Elemente *hWnd:* Bezeichnet das Fenster, das die Botschaft empfängt.

wMessage: Gibt den Botschaftsbezeichner an.

wParam: Bestimmt Zusatzinformationen über die Botschaft. Die genaue Bedeutung ist von *wMessage* abhängig.

lParam: Bestimmt Zusatzinformationen über die Botschaft. Die genaue Bedeutung ist von *wMessage* abhängig.

HELPWININFO 3.1

Definition

```
typedef struct {
   int wStructSize;
   int x;
   int y;
   int dx;
   int dy;
   int wMax;
   char rgchMember[2];
} HELPWININFO;
```

Beschreibung Die *HELPWININFO*-Struktur enthält die Größe und die Position eines sekundären Hilfe-Fensters. Eine Anwendung kann diese Größe durch einen Aufruf der Funktion *WinHelp* mit dem Wert *HELP_SETWINPOS* festlegen.

Elemente *wStructSize:* Gibt die Größe der *HELPWININFO*-Struktur an.

x: Gibt die x-Koordinate der oberen linken Ecke des Fensters an.

y: Gibt die y-Koordinate der oberen linken Ecke des Fensters an.

dx: Bestimmt die Breite des Fensters.

dy: Bestimmt die Höhe des Fensters.

wMax: Legt fest, ob das Fenster vergrößert oder mit der angegebenen Position und Abmessung dargestellt werden soll. Wenn dieser Wert 1 ist, wird das Fenster vergrößert. Wenn er 0 ist, werden Größe und Position von den Elementen *x*, *y*, *dx* und *dy* bestimmt.

rgchMember: Gibt den Namen des Fensters an.

Hinweis Die Windows-Hilfe unterteilt den Bildschirm sowohl in x- als auch in y-Richtung in 1024 Einheiten. Um ein sekundäres Fenster zu erzeugen, das den oberen linken Quadranten des Bildschirms ausfüllt, müßte eine Anwendung für die Elemente *x* und *y* 0 und für die Elemente *dx* und *dy* 512 angeben.

Siehe auch *WinHelp*

HSZPAIR 3.1

Definition

```
# include <ddeml.h>

typedef struct tagHSZPAIR { /* hp */
   HSZ hszSvc;
   HSZ hszTopic;
} HSZPAIR;
```

Beschreibung Die *HSZPAIR*-Struktur enthält den Service- und Themennamen für einen dynamischen Datenaustausch (DDE). Eine DDE-Server-Anwendung kann diese Struktur während einer *XTYP_WILDCONNECT*-Transaktion zur Aufzählung der Service-/Themenname-Paare verwenden, die sie unterstützt.

Elemente *hszSvc:* Bezeichnet einen Servicenamen.

hszTopic: Bezeichnet einen Themennamen.

KERNINGPAIR 3.1

Definition

```
typedef struct tagKERNINGPAIR {
   WORD wFirst;
   WORD wSecond;
   int  iKernAmount;
} KERNINGPAIR;
```

Beschreibung Die *KERNINGPAIR*-Struktur definiert ein Unterschnittpaar.

Elemente *wFirst:* Gibt den Zeichencode für das erste Zeichen im Unterschnittpaar an.

wSecond: Gibt den Zeichencode für das zweite Zeichen im Unterschnittpaar an.

iKernAmount: Bestimmt den Betrag, um den die Zeichen dieses Paares unterschnitten werden, wenn sie nebeneinander in der gleichen Schrift und Größe erscheinen. Dieser Wert ist normalerweise negativ, weil das Unterschneiden üblicherweise dazu führt, daß zwei Zeichen enger als normal nebeneinander gesetzt werden. Der Wert wird in virtuellen Einheiten angegeben, er ist also vom aktuellen Abbildungsmodus abhängig.

Siehe auch *GetKerningPairs*

LOCALENTRY 3.1

Definition

```
#include <toolhelp.h>

typedef struct tagLOCALENTRY { /* le */
   DWORD dwSize;
   HLOCAL hHandle;
```

```
    WORD wAddress;
    WORD wSize;
    WORD wFlags;
    WORD wcLock;
    WORD wType;
    WORD hHeap;
    WORD wHeapType;
    WORD wNext;
} LOCALENTRY;
```

Beschreibung Die *LOCALENTRY*-Struktur enthält Informationen über ein Speicherobjekt auf dem lokalen Heap.

Elemente *dwSize:* Gibt die Größe der *LOCALENTRY*-Struktur in Byte an.

hHandle: Bezeichnet das lokale Speicherobjekt.

wAddress: Gibt die Adresse des lokalen Speicherobjekts an.

wSize: Gibt die Größe des lokalen Speicherobjekts in Byte an.

wFlags: Gibt an, ob das Speicherobjekt fest, frei oder verschiebbar ist. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
LF_FIXED	Das Objekt befindet sich an einer festen Speicherposition.
LF_FREE	Das Objekt ist Teil des freien Speicherbereichs.
LF_MOVABLE	Das Objekt kann verschoben werden, um Speicher zu verdichten.

wcLock: Gibt den Sperrenzähler an. Wenn dieser Wert Null ist, ist das Objekt nicht gesperrt.

wType: Gibt den Inhalt des Speicherobjekts an. Dieses Element kann einen der folgenden Werte annehmen.

Wert	Bedeutung
LT_FREE	Das Objekt gehört zum freien Speicherbereich.
LT_GDI_BITMAP	Das Objekt enthält einen Bitmap-Header.
LT_GDI_BRUSH	Das Objekt enthält einen Pinsel.
LT_GDI_DC	Das Objekt enthält einen Gerätekontext.
LT_GDI-DISABLED_DC	Das Objekt ist zur internen Verwendung durch Windows reserviert.

Wert	Bedeutung
LT_GDI_FONT	Das Objekt enthält einen Schrift-Header.
LT_GDI_MAX	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_GDI_METADC	Das Objekt enthält einen Metafile-Gerätekontext.
LT_GDI_METAFILE	Das Objekt enthält einen Metafile-Header.
LT_GDI_PALETTE	Das Objekt enthält eine Palette.
LT_GDI_PEN	Das Objekt enthält einen Stift.
LT_GDI_RGN	Das Objekt enthält eine Region.
LT_NORMAL	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_USER_ATOMS	Das Objekt enthält eine Atomstruktur.
LT_USER_BWL	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_USER_CBOX	Das Objekt enthält eine Kombinationsfensterstruktur.
LT_USER_CHECKPOINT	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_USER_CLASS	Das Objekt enthält eine Klassenstruktur.
LT_USER_CLIP	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_USER_DCE	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_USER_ED	Das Objekt enthält eine Editierfeldstruktur.
LT_USER_HANDLETABLE	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_USER_HOOKLIST	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_USER_HOTKEYLIST	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_USER_LBIV	Das Objekt enthält eine Listenstruktur.
LT_USER_LOCKINPUTSTATE	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_USER_MENU	Das Objekt enthält eine Menüstruktur.
LT_USER_MISC	Das Objekt ist zur internen Verwendung durch Windows reserviert.

Wert	Bedeutung
LT_USER_MWP	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_USER_OWNER-DRAW	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_USER_PALETTE	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_USER_POPUP-MENU	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_USER_PROP	Das Objekt enthält eine Fenstereigenschafts-struktur.
LT_USER_SPB	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_USER_STRING	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_USER_USERSEE-USERDOALLOC	Das Objekt ist zur internen Verwendung durch Windows reserviert.
LT_USER_WND	Das Objekt enthält eine Fensterstruktur.

hHeap: Bezeichnet den lokalen Speicher-Heap.

wHeapType: Bestimmt den Typ des lokalen Heap. Dieses Element kann einen der folgenden Werte enthalten:

Wert	Bedeutung
NORMAL_HEAP	Der Heap ist der Standard-Heap.
USER_HEAP	Der Heap wird vom USER-Modul verwendet.
GDI_HEAP	Der Heap wird vom GDI-Modul verwendet.

wNext: Gibt das nächste Element im lokalen Heap an. Dieses Element ist zur internen Verwendung durch Windows reserviert.

Hinweis

Die Werte für *wType* dienen nur Informationszwecken. Microsoft behält sich vor, diese Werte jederzeit zu ändern oder zu entfernen. Anwendungen sollten niemals Elemente auf den System-Heaps direkt ändern, weil sich diese Informationen in zukünftigen Versionen ändern können. Die Werte von *wType* für das USER-Modul sind nur in den Debug-Versionen von USER.EXE enthalten.

Siehe auch

LocalFirst, *LocalNext*, *LOCALINFO*

LOCALINFO 3.1

Definition

```
#include <toolhelp.h>

typedef struct tagLOCALINFO { /* li */
    DWORD dwSize;
    WORD  wcItems;
} LOCALINFO;
```

Beschreibung Die *LOCALINFO*-Struktur enthält Informationen über den lokalen Heap.

Elemente *dwSize:* Gibt die Größe der *LOCALINFO*-Struktur in Byte an.

wcItems: Gibt die Gesamtanzahl der Elemente auf dem lokalen Heap an.

Siehe auch *LocalInfo, LOCALENTRY*

MAT2 3.1

Definition

```
typedef struct tagMAT2 { /* mat2 */
    FIXED eM11;
    FIXED eM12;
    FIXED eM21;
    FIXED eM22;
} MAT2;
```

Beschreibung Die *MAT2*-Struktur enthält die Werte für eine Transformationsmatrix.

Elemente *eM11:* Bestimmt einen Festkommawert für das Element *M11* einer 2 x 2 Transformationsmatrix.

eM12: Bestimmt einen Festkommawert für das Element *M12* einer 2 x 2 Transformationsmatrix.

eM21: Bestimmt einen Festkommawert für das Element *M21* einer 2 x 2 Transformationsmatrix.

eM22: Bestimmt einen Festkommawert für das Element *M22* einer 2 x 2 Transformationsmatrix.

Hinweis Die Einheitsmatrix erzeugt eine Transformation, in der das transformierte grafische Objekt identisch mit dem Quellobjekt ist. In der Einheitsmatrix ist der Wert von *M11* 1, der von *eM12* Null, *eM21* hat den Wert Null und *eM22* den Wert 1.

Siehe auch *GetGlyphOutline*

MEMMANINFO 3.1

Definition

```
#include <toolhelp.h>

typedef struct tagMEMMANINFO { /* mmi */
    DWORD dwSize;
    DWORD dwLargestFreeBlock;
    DWORD dwMaxPagesAvailable;
    DWORD dwMaxPagesLockable;
    DWORD dwTotalLinearSpace;
    DWORD dwTotalUnlockedPages;
    DWORD dwFreePages;
    DWORD dwTotalPages;
    DWORD dwFreeLinearSpace;
    DWORD dwSwapFilePages;
    WORD  wPageSize;
} MEMMANINFO;
```

Beschreibung Die *MEMMANINFO*-Struktur enthält Informationen über den Status und die Leistung der virtuellen Speicherverwaltung. Wenn die Speicherverwaltung im Standardmodus läuft, ist nur das Element *dwLargestFreeBlock* dieser Struktur gültig.

Elemente *dwSize:* Gibt die Größe der *MEMMANINFO*-Struktur in Byte an.

dwLargestFreeBlock: Gibt den größten freien Block fortlaufenden linearen Speichers im System in Byte an.

dwMaxPagesAvailable: Gibt die maximale Anzahl von Speicherseiten an, die im System reserviert werden können: der Wert von *dwLargestFreeBlock* dividiert durch den Wert von *wPageSize*.

dwMaxPagesLockable: Gibt die maximale Anzahl von Speicherseiten an, die reserviert und gesperrt werden können.

dwTotalLinearSpace: Gibt die Größe des gesamten linearen Adreßraums in Seiten an.

dwTotalUnlockedPages: Gibt die Anzahl der ungesperrten Seiten im System an. Dieser Wert schließt die freien Seiten ein.

dwFreePages: Gibt die Anzahl der Seiten an, die nicht in Gebrauch sind.

dwTotalPages: Gibt die Gesamtzahl der Seiten an, die die virtuelle Speicherverwaltung verwaltet. Dieser Wert schließt freie, gesperrte und ungesperrte Speicherseiten ein.

dwFreeLinearSpace: Gibt die Größe des freien Speichers im linearen Adreßraum in Seiten an.

dwSwapFilePages: Gibt die Anzahl der Seiten in der Swap-Datei des Systems an.

wPageSize: Gibt die Größe einer System-Speicherseite in Byte an.

Siehe auch *MemManInfo*

METAHEADER 3.1

Definition

```
typedef struct tagMETAHEADER { /* mh */
    UINT mtType;
    UINT mtHeaderSize;
    UINT mtVersion;
    DWORD mtSize;
    UINT mtNoObjects;
    DWORD mtMaxRecord;
    UINT mtNoParameters;
} METAHEADER;
```

Beschreibung Die *METAHEADER*-Struktur enthält Informationen über ein Metafile.

Elemente *mtType:* Bestimmt, ob das Metafile im Speicher oder in einer Datei aufgezeichnet ist. Dieses Element kann einen der beiden folgenden Werte annehmen:

Wert	Bedeutung
1	Das Metafile ist im Speicher.
2	Das Metafile ist in einer Datei.

mtHeaderSize: Gibt die Größe des Metafile-Headers in Einheiten vom Typ *WORD* an.

mtVersion: Gibt die Windows-Versionsnummer an. Die Versionsnummer für Metafiles, die geräteunabhängige Bitmaps (DIBs) unterstützen, ist 0x0300. Andernfalls ist die Versionsnummer 0x0100.

mtSize: Gibt die Größe der Datei in Einheiten vom Typ *WORD* an.

mtNoObjects: Gibt die maximale Anzahl der Objekte an, die gleichzeitig im Metafile vorhanden sind.

mtMaxRecord: Gibt die Größe des größten Record im Metafile in Einheiten vom Typ *WORD* an.

mtNoParameters: Reserviert.

Siehe auch *METARECORD*

METARECORD 3.1

Definition

```
typedef struct tagMETARECORD { /* mr */
    DWORD rdSize;
    UINT rdFunction;
    UINT rdParm[1];
} METARECORD;
```

Beschreibung Die *METARECORD*-Struktur enthält ein Metafile-Record.

Elemente *rdSize:* Gibt die Größe des Record in Einheiten vom Typ *WORD* an.

rdFunction: Bestimmt die Funktionsnummer.

rdParm: Gibt ein Array mit Werten vom Typ *WORD* an, die die Funktionsparameter in der umgekehrten Reihenfolge enthalten, in der sie an die Funktion übergeben werden.

Siehe auch *METAHEADER*

MINMAXINFO 3.1

Definition

```
typedef struct tagMINMAXINFO { /* mmi */
    POINT ptReserved;
    POINT ptMaxSize;
    POINT ptMaxPosition;
    POINT ptMinTrackSize;
    POINT ptMaxTrackSize;
} MINMAXINFO;
```

Beschreibung Die *MINMAXINFO*-Struktur enthält Informationen über die Größe und Position eines vergrößerten Fensters und seine minimale und maximale Ausdehnung, die mit Hilfe der Maus zu erreichen ist.

Elemente *ptReserved:* Zur internen Verwendung reserviert.

ptMaxSize: Bestimmt die maximierte Breite (*point.x*) und Höhe (*point.y*) des Fensters.

ptMaxPosition: Legt die Position der linken Seite (*point.x*) und der oberen Seite (*point.y*) des vergrößerten Fensters fest.

ptMinTrackSize: Bestimmt die minimale Breite (*point.x*) und Höhe (*point.y*) des Fensters, die mit der Maus zu erreichen ist.

ptMaxTrackSize: Bestimmt die maximale Breite (*point.x*) und Höhe (*point.y*) des Fensters, die mit der Maus zu erreichen ist.

Siehe auch *POINT*, *WM_GETMINMAXINFO*

MODULEENTRY 3.1

Definition

```
#include <toolhelp.h>

typedef struct tagMODULEENTRY { /* me */
    DWORD dwSize;
    char szModule[MAX_MODULE_NAME + 1];
    HMODULE hModule;
    WORD wcUsage;
    char szExePath[MAX_PATH + 1];
    WORD wNext;
} MODULEENTRY;
```

Beschreibung Die *MODULEENTRY*-Struktur enthält Informationen über ein Modul in der Modulliste.

Elemente *dwSize:* Gibt die Größe der *MODULEENTRY*-Struktur in Byte an.

szModule: Gibt den nullterminierten String an, der den Modulnamen enthält.

hModule: Bezeichnet das Modul-Handle.

wcUsage: Gibt den Referenzzähler des Moduls an. Diese Zahl wird auch von der Funktion *GetModuleUsage* zurückgegeben.

szExePath: Gibt den nullterminierten String an, der den vollständigen Pfad der ausführbaren Datei für das Modul enthält.

wNext: Gibt das nächste Modul in der Modulliste an. Dieses Element ist zur internen Verwendung von Windows reserviert.

Siehe auch *ModuleFindHandle, ModuleFindName, ModuleFirst, ModuleNext*

MONCBSTRUCT 3.1

Definition

```
#include <ddeml.h>

typedef struct tagMONCBSTRUCT { /* mcbst */
   UINT cb;
   WORD wReserved;
   DWORD dwTime;
   HANDLE hTask;
   DWORD dwRet;
   UINT wType;
   UINT wFmt;
   HCONV hConv;
   HSZ hsz1;
   HSZ hsz2;
   HDDEDATA hData;
   DWORD dwData1;
   DWORD dwData2;
} MONCBSTRUCT;
```

Beschreibung Die *MONCBSTRUCT*-Struktur enthält Informationen über die aktuelle DDE-Transaktion. Eine DDE-überwachende Anwendung kann diese Struktur beim Überwachen von Transaktionen verwenden, die das System an die DDE-Callback-Funktionen anderer Anwendungen übergibt.

Elemente *cb:* Gibt die Größe der Struktur in Byte an.

wReserved: Reserviert.

dwTime: Gibt die Windows-Zeit an, zu der die Transaktion stattgefunden hat. Die Windows-Zeit ist die Anzahl der Millisekunden seit dem Start des Systems.

hTask: Bezeichnet die Task (Anwendungsinstanz), die die DDE-Callback-Funktion enthält, die die Transaktion empfangen hat.

dwRet: Gibt den Rückgabewert der DDE-Callback-Funktion an, die die Transaktion bearbeitet hat.

wType: Bestimmt den Transaktionstyp.

wFmt: Bestimmt das Format der Daten (falls vorhanden), die während der Transaktion ausgetauscht wurden.

hConv: Bezeichnet die Konversation, in der die Transaktion stattgefunden hat.

hsz1: Bezeichnet einen String.

hsz2: Bezeichnet einen String.

hData: Bezeichnet die Daten (falls vorhanden), die während der Transaktion ausgetauscht wurden.

dwData1: Bestimmt zusätzliche Daten.

dwData2: Bestimmt zusätzliche Daten.

Siehe auch *MONERRSTRUCT*, *MONHSZSTRUCT*, *MONLINKSTRUCT*, *MONMSGSTRUCT*

MONCONVSTRUCT 3.1

Definition

```
#include <ddeml.h>

typedef struct tagMONCONVSTRUCT { /* mcvst */
    UINT cb;
    BOOL fConnect;
    DWORD dwTime;
    HANDLE hTask;
    HSZ hszSvc;
    HSZ hszTopic;
    HCONV hConvClient;
```

```
    HCONV hConvServer;
} MONCONVSTRUCT;
```

Beschreibung Die *MONCONVSTRUCT*-Struktur enthält Informationen über eine Konversation. Eine Anwendung, die den dynamischen Datenaustausch (DDE) überwacht, kann diese Struktur verwenden, um Informationen über eine eingerichtete oder beendete Advise-Schleife anzufordern.

Elemente *cb:* Gibt die Größe der Struktur in Byte an.

fConnect: Zeigt an, ob die Konversation aktuell eingerichtet ist. Ein Wert von *TRUE* zeigt an, daß die Konversation eingerichtet, *FALSE*, daß sie es nicht ist.

dwTime: Gibt die Windows-Zeit an, zu der die Konversation begonnen oder beendet wurde. Die Windows-Zeit ist die Anzahl der Millisekunden seit dem Start des Systems.

hTask: Bezeichnet eine Task (Anwendungsinstanz), die ein Partner in der Konversation ist.

hszSvc: Bezeichnet den Servicenamen, für den die Konversation eingerichtet ist.

hszTopic: Bezeichnet den Themennamen, für den die Konversation eingerichtet ist.

hConvClient: Bezeichnet die Client-Konversation.

hConvServer: Bezeichnet die Server-Konversation.

Siehe auch *MONCBSTRUCT*, *MONERRSTRUCT*, *MONHSZSTRUCT*, *MONLINKSTRUCT*, *MONMSGSTRUCT*

MONERRSTRUCT 3.1

Definition

```
#include <ddeml.h>

typedef struct tagMONERRSTRUCT { /* mest */
    UINT cb;
    UINT wLastError;
    DWORD dwTime;
    HANDLE hTask;
} MONERRSTRUCT;
```

Beschreibung Die *MONERRSTRUCT*-Struktur enthält Informationen über den aktuellen DDE-Fehler. Eine DDE-überwachende Anwendung kann diese Struktur zur Überwachung von Fehlern verwenden, die von den Funktionen der DDE-Verwaltungsbibliothek (DDEML) zurückgegeben werden.

Elemente *cb:* Gibt die Größe der Struktur in Byte an.

wLastError: Bestimmt den aktuellen Fehler.

dwTime: Gibt die Windows-Zeit an, zu der der Fehler aufgetreten ist. Die Windows-Zeit ist die Anzahl der Millisekunden seit dem Start des Systems.

hTask: Bezeichnet die Task (Anwendungsinstanz), die die den Fehler verursachende DDE-Funktion aufgerufen hat.

Siehe auch *MONCBSTRUCT*, *MONCONVSTRUCT*, *MONHSZSTRUCT*, *MONLINKSTRUCT*, *MONMSGSTRUCT*

MONHSZSTRUCT 3.1

Definition

```
#include <ddeml.h>

typedef struct tagMONHSZSTRUCT { /* mhst */
    UINT cb;
    BOOL fsAction;              /* MH_-Wert */
    DWORD dwTime;
    HSZ hsz;
    HANDLE hTask;
    WORD wReserved;
    char str[1];
} MONHSZSTRUCT;
```

Beschreibung Die *MONHSZSTRUCT*-Struktur enthält Informationen über ein DDE-String-Handle. Eine DDE-überwachende Anwendung kann diese Struktur beim Überwachen der String-Verwaltung der DDE-Verwaltungsbibliothek (DDEML) verwenden.

Elemente *cb:* Gibt die Größe der Struktur in Byte an.

fsAction: Gibt die Aktion an, die mit dem durch das Element *hsz* bezeichneten String-Handle durchgeführt wird.

Das Element *fsAction* kann einen der folgenden Werte annehmen:

Wert	Bedeutung
MH_CLEANUP	Eine Anwendung gibt ihre DDE-Ressourcen frei und veranlaßt das System, die von ihr erzeugten String-Handles zu löschen (die Anwendung hat die Funktion *DdeUninitialize* aufgerufen).
MH_CREATE	Eine Anwendung erzeugt ein String-Handle (die Anwendung hat die Funktion *DdeCreateStringHandle* aufgerufen).
MH_DELETE	Eine Anwendung löscht ein String-Handle (die Anwendung hat die Funktion *DdeFreeStringHandle* aufgerufen).

Wert	Bedeutung
MH_KEEP	Eine Anwendung erhöht den Verwendungszähler eines String-Handles (die Anwendung hat die Funktion *DdeKeepStringHandle* aufgerufen).

dwTime: Gibt die Windows-Zeit an, zu der die vom Element *fsAction* angegebene Aktion stattgefunden hat. Die Windows-Zeit ist die Anzahl der Millisekunden seit dem Start des Systems.

hsz: Bezeichnet den String.

hTask: Bezeichnet die Task (Anwendungsinstanz), die die Aktion mit dem String-Handle durchführt.

wReserved: Reserviert.

str: Zeigt auf den String, der vom Element *hsz* bezeichnet wird.

Siehe auch *MONCBSTRUCT*, *MONCONVSTRUCT*, *MONERRSTRUCT*, *MONLINKSTRUCT*, *MONMSGSTRUCT*

MONLINKSTRUCT 3.1

Definition

```
#include <ddeml.h>
typedef struct tagMONLINKSTRUCT { /* mlst */
    UINT cb;
    DWORD dwTime;
    HANDLE hTask;
    BOOL fEstablished;
    BOOL fNoData;
    HSZ hszSvc;
    HSZ hszTopic;
    HSZ hszItem;
    UINT wFmt;
    BOOL fServer;
    HCONV hConvServer;
    HCONV hConvClient;
} MONLINKSTRUCT;
```

Beschreibung Die *MONLINKSTRUCT*-Struktur enthält Informationen über eine DDE-Advise-Schleife. Eine DDE-überwachende Anwendung kann diese Struktur verwenden, um Informationen über eine gestartete oder beendete Advise-Schleife anzufordern.

Elemente *cb:* Gibt die Größe der Struktur in Byte an.

dwTime: Gibt die Windows-Zeit an, zu der die Advise-Schleife gestartet oder beendet wurde. Die Windows-Zeit ist die Anzahl der Millisekunden seit dem Start des Systems.

hTask: Bezeichnet die Task (Anwendungsinstanz), die ein Partner in der Advise-Schleife ist.

fEstablished: Zeigt an, ob eine Advise-Schleife erfolgreich eingerichtet wurde. Ein Wert von *TRUE* zeigt an, daß eine Advise-Schleife eingerichtet, *FALSE*, daß sie nicht eingerichtet wurde.

fNoData: Zeigt an, ob das *XTYPF_NODATA*-Flag für die Advise-Schleife gesetzt worden ist. Ein Wert von *TRUE* zeigt an, daß das Flag gesetzt, *FALSE*, daß es nicht gesetzt ist.

hszSvc: Bezeichnet den Servicenamen der Server-Anwendung in der Advise-Schleife.

hszTopic: Bezeichnet den Themennamen, für den die Advise-Schleife eingerichtet ist.

hszItem: Bezeichnet den Elementnamen, der Gegenstand der Advise-Schleife ist.

wFmt: Bestimmt das Format der während der Advise-Schleife ausgetauschten Daten (falls vorhanden).

fServer: Zeigt an, ob die Verknüpfungsbenachrichtigung vom Server gekommen ist. Wenn das der Fall ist, ist dieser Wert *TRUE*, andernfalls *FALSE*.

hConvServer: Bezeichnet die Server-Konversation.

hConvClient: Bezeichnet die Client-Konversation.

Siehe auch *MONCBSTRUCT*, *MONERRSTRUCT*, *MONHSZSTRUCT*, *MONMSGSTRUCT*

MONMSGSTRUCT 3.1

Definition

```
#include <ddeml.h>

typedef struct tagMONMSGSTRUCT { /* mmst */
    UINT cb;
    HWND hwndTo;
    DWORD dwTime;
    HANDLE hTask;
    UINT wMsg;
    WPARAM wParam;
    LPARAM lParam;
} MONMSGSTRUCT;
```

Beschreibung Die *MONMSGSTRUCT*-Struktur enthält Informationen über eine DDE-Botschaft. Eine DDE-überwachende Anwendung kann diese Struktur verwenden, um Informationen über eine gesendete DDE-Botschaft zu ermitteln.

Elemente *cb:* Gibt die Größe der Struktur in Byte an.

hwndTo: Bezeichnet das Fenster, das die DDE-Botschaft empfängt.

dwTime: Gibt die Windows-Zeit an, zu der die Botschaft gesendet wurde. Die Windows-Zeit ist die Anzahl der Millisekunden seit dem Start des Systems.

hTask: Bezeichnet die Task (Anwendungsinstanz), die das Fenster enthält, das die DDE-Botschaft empfängt.

wMsg: Gibt den Bezeichner der DDE-Botschaft an.

wParam: Gibt den Parameter *wParam* der DDE-Botschaft an.

lParam: Gibt den Parameter *lParam* der DDE-Botschaft an.

Siehe auch *MONCBSTRUCT*, *MONCONVSTRUCT*, *MONERRSTRUCT*, *MONHSZSTRUCT*, *MONLINKSTRUCT*

MOUSEHOOKSTRUCT 3.1

Definition

```
typedef struct tagMOUSEHOOKSTRUCT { /* ms */
    POINT pt;
    HWND hwnd;
    UINT wHitTestCode;
    DWORD dwExtraInfo;
} MOUSEHOOKSTRUCT;
```

Beschreibung Die *MOUSEHOOKSTRUCT*-Struktur enthält Informationen über ein Mausereignis.

Elemente *pt:* Gibt eine *POINT*-Struktur an, die die x- und y-Koordinaten des Mauszeigers in Bildschirmkoordinaten enthält.

hwnd: Bezeichnet das Fenster, das die dem Mausereignis entsprechende Mausbotschaft erhält.

wHitTestCode: Gibt den Code für den Treffertest an.

dwExtraInfo: Gibt zusätzliche Informationen über das Mausereignis an. Eine Anwendung kann diese Informationen durch die Funktion *GetMessageExtraInfo* abrufen.

Siehe auch *GetMessageExtraInfo*, *SetWindowsHook*

NCCALCSIZE_PARAMS 3.1

Definition

```
typedef struct tagNCCALCSIZE_PARAMS {
    RECT          rgrc[3];
    WINDOWPOS FAR* lppos;
} NCCALCSIZE_PARAMS;
```

Beschreibung Die *NCCALCSIZE_PARAMS*-Struktur enthält Informationen, die eine Anwendung während der Bearbeitung der *WM_*-

NCCALCSIZE-Botschaft verwenden kann, um die Größe, Position und den gültigen Inhalt des Client-Bereichs eines Fensters zu errechnen.

Elemente

rgrc: Gibt ein Array an, das Rechtecke enthält. Das erste enthält die neuen Koordinaten eines Fensters, das verschoben oder dessen Größe geändert worden ist. Das zweite enthält die Koordinaten des Fensters vor der Verschiebung oder Größenänderung. Das dritte enthält die Koordinaten des Client-Bereichs eines Fensters, bevor es verschoben oder in der Größe geändert worden ist. Wenn das Fenster untergeordnet ist, sind die Koordinaten relativ zum Client-Bereich des übergeordneten Fensters. Wenn es ein Fenster oberster Ebene ist, sind die Koordinaten relativ zum Bildschirm.

lppos: Zeigt auf eine *WINDOWPOS*-Struktur, die die Werte der Größen- und Positionsveränderung enthält. Sie hat folgende Form:

```
typedef struct tagWINDOWPOS { /*wp */
    HWND hwnd;
    HWND hwndInsertAfter;
    int x;
    int y;
    int cx;
    int cy;
    UINT flags;
} WINDOWPOS;
```

Siehe auch

MoveWindow, SetWindowPos, RECTWINDOWPOS, WM_NCCALCSIZE

NEWCPLINFO 3.1

Definition

```
#include <cpl.h>

typedef struct tagNEWCPLINFO { /* ncpli              */
  DWORD dwSize;         /* Größe der Struktur        */
  DWORD dwFlags;
  DWORD dwHelpContext;  /* Hilfekontext              */
  LONG lData;           /* benutzerdefinierte Daten  */
  HICON hIcon;          /* Symbol, gehört zu CONTROL.EXE
                           (kann gelöscht werden)    */
  char szName[32];      /* kurzer Name               */
  char szInfo[64];      /* langer Name (Statuszeile) */
  char szHelpFile[128]; /* Pfad auf die Hilfedatei   */
} NEWCPLINFO;
```

Beschreibung

Die *NEWCPLINFO*-Struktur enthält Ressourceninformationen und einen benutzerdefinierten Wert für eine Anwendung der Systemsteuerung.

Elemente

dwSize: Gibt die Größe der Struktur in Byte an.

dwFlags: Gibt Flags für die Systemsteuerung an.

dwHelpContext: Bestimmt die Kontextnummer für das anzuzeigende Thema in der Projektdatei (.HPJ) der Hilfe-Anwendung. Dieses Thema wird angezeigt, wenn der Benutzer Hilfe für die Anwendung anfordert.

lData: Bestimmt die anwendungsdefinierten Daten.

hIcon: Bezeichnet eine Symbolressource für das Anwendungssymbol. Dieses Symbol wird im Fenster *Systemsteuerung* angezeigt.

szName: Gibt einen nullterminierten String an, der den Namen der Anwendung enthält. Dieser Name ist der kurze String, der unterhalb des Anwendungssymbols im Fenster *Systemsteuerung* angezeigt wird. Der Name wird auch im Menü *Einstellungen* der Systemsteuerung angezeigt.

szInfo: Gibt einen nullterminierten String an, der die Beschreibung der Anwendung enthält. Sie wird am unteren Rand des Fensters *Systemsteuerung* angezeigt, wenn das Anwendungssymbol ausgewählt wird.

szHelpFile: Gibt einen nullterminierten String an, der den Pfad der Hilfedatei enthält, falls eine solche für die Anwendung existiert.

NEWTEXTMETRIC 2.x

Definition

```
typedef struct tagNEWTEXTMETRIC { /* ntm */
    int tmHeight;
    int tmAscent;
    int tmDescent;
    int tmInternalLeading;
    int tmExternalLeading;
    int tmAveCharWidth;
    int tmMaxCharWidth;
    int tmWeight;
    BYTE tmItalic;
    BYTE tmUnderlined;
    BYTE tmStruckOut;
    BYTE tmFirstChar;
    BYTE tmLastChar;
    BYTE tmDefaultChar;
    BYTE tmBreakChar;
    BYTE tmPitchAndFamily;
    BYTE tmCharSet;
    int tmOverhang;
```

```
    int tmDigitizedAspectX;
    int tmDigitizedAspectY;
    DWORD ntmFlags;      /* verschiedene Flags        */
    UINT ntmSizeEM;      /* Größe von EM              */
    UINT ntmCellHeight;  /* Schrifthöhe in
                            fiktiven Einheiten        */
    UINT ntmAvgWidth;    /* durchschnittliche Breite
                            in fiktiven Einheiten     */
} NEWTEXTMETRIC;
```

Beschreibung Die *NEWTEXTMETRIC*-Struktur enthält grundlegende Informationen über eine physikalische Schrift. Die letzten vier Elemente dieser Struktur sind nicht in der *TEXTMECTRIC*-Struktur enthalten, in jeder anderen Hinsicht sind diese beiden Strukturen jedoch identisch. Die zusätzlichen Elemente werden für Informationen über TrueType-Schriften verwendet.

Elemente *tmHeight:* Gibt die Höhe der Zeichenzellen an. Die Höhe ist die Summe der Elemente *tmAscent* und *tmDescent*.

tmAscent: Gibt die Oberlänge der Zeichenzellen an. Die Oberlänge ist der Abstand zwischen dem oberen Rand der Zeichenzelle und der Grundlinie.

tmDescent: Gibt die Unterlänge der Zeichenzellen an. Die Unterlänge ist der Abstand zwischen dem unteren Rand der Zeichenzelle und der Grundlinie.

tmInternalLeading: Gibt die Differenz zwischen der Punktgröße einer Schrift und der physikalischen Größe der Schrift an. Bei TrueType-Schriften beträgt der Wert *tmHeight* minus (*s* * *ntmSizeEM*), wobei *s* der Skalierungsfaktor der TrueType-Schrift ist. Bei Bitmap-Schriften wird dieser Wert verwendet, um die Punktgröße einer Schrift zu ermitteln. Wenn eine Anwendung einen negativen Wert im Element *lfHeight* der *LOGFONT*-Struktur angibt, fordert sie eine Schrift an, deren Höhe *tmHeight* minus *tmInternalLeading* ist.

tmExternalLeading: Bestimmt den Betrag des zusätzlichen Leerraums, den eine Anwendung zwischen Spalten einfügt. Da dieser Bereich außerhalb der Zeichenzellen liegt, enthält er keine Markierungen und wird weder durch Textausgabeaufrufe im transparenten noch im undurchsichtigen Modus verändert. Der Schrift-Designer setzt dieses Element manchmal auf Null.

tmAveCharWidth: Bestimmt die durchschnittliche Zeichenbreite einer Schrift. Bei *ANSI_CHARSET*-Schriften ist dieser Wert der gewichtete Durchschnitt der Zeichen »a« bis »z« und des Leerzeichens. Bei anderen Zeichensätzen ist er ein ungewichteter Durchschnitt aller Zeichen der Schrift.

tmMaxCharWidth: Gibt die Breite des breitesten Zeichens in der Schrift an.

tmWeight: Bestimmt die Stärke der Schrift.

Dieses Element kann einen der folgenden Werte annehmen:

Konstante	Wert
FW_DONTCARE	0
FW_THIN	100
FW_EXTRALIGHT	200
FW_ULTRALIGHT	200
FW_LIGHT	300
FW_NORMAL	400
FW_REGULAR	400
FW_MEDIUM	500
FW_SEMIBOLD	600
FW_DEMIBOLD	600
FW_BOLD	700
FW_EXTRABOLD	800
FW_ULTRABOLD	800
FW_BLACK	900
FW_HEAVY	900

tmItalic: Legt eine kursive Schrift fest, wenn es ungleich Null ist.

tmUnderlined: Legt eine unterstrichene Schrift fest, wenn es ungleich Null ist.

tmStruckOut: Legt eine durchgestrichene Schrift fest, wenn es ungleich Null ist.

tmFirstChar: Gibt den Wert des ersten Zeichens an, das in der Schrift definiert ist.

tmLastChar: Gibt den Wert des letzten Zeichens an, das in der Schrift definiert ist.

tmDefaultChar: Bestimmt den Wert des Zeichens, das für Zeichen verwendet wird, die nicht in der Schrift enthalten sind.

tmBreakChar: Gibt den Wert des Zeichens an, das zur Definition von Wortumbrüchen für die Textausrichtung verwendet wird.

tmPitchAndFamily: Gibt den Zeichenabstand und die Familie der gewählten Schrift an. Die vier niederwertigen Bits bezeichnen den Schrifttyp. Sie können die folgenden Werte annehmen:

Wert	Bedeutung
TMPF_PITCH	Bestimmt eine Schrift mit festem Zeichenabstand.
TMPF_VECTOR	Bestimmt eine Vektor- oder eine TrueType-Schrift.
TMPF_TRUETYPE	Bestimmt eine TrueType-Schrift.
TMPF_DEVICE	Bestimmt eine Geräteschrift.

Einige Schriften werden von mehreren dieser Bits bezeichnet. Für die TrueType-Schrift Courier New mit festem Zeichenabstand würden die Bits *TMPF_PITCH*, *TMPF_VECTOR* und *TMPF_TRUETYPE* gesetzt werden.

Wenn das Bit *TMPF_TRUETYPE* gesetzt ist, kann die Schrift auf allen Ausgabegeräten verwendet werden. Wenn z.B. eine TrueType-Schrift auf einem Drucker vorhanden wäre, aber nicht für die Bildschirmanzeige verwendet werden könnte, würde das Bit *TMPF_TRUETYPE* für diese Schrift nicht gesetzt werden.

Die vier höherwertigen Bits bestimmen die Schriftfamilie. Das Element *tmPitchAndFamily* kann mit dem hexadezimalen Wert 0xF0 durch den bitweisen UND-Operator kombiniert werden und dann mit den Schriftfamiliennamen verglichen werden, um eine Entsprechung zu finden. Die folgenden Schriftfamilien sind definiert:

Wert	Bedeutung
FF_DECORATIVE	Schmuckschriften, z.B. Alt Englisch.
FF_DONTCARE	Nicht wichtig oder unbekannt.
FF_MODERN	Schriften mit konstantem Zeichenabstand, mit oder ohne Serifen. Beispiele hierfür sind Pica, Elite und Courier New.
FF_ROMAN	Schriften mit variablem Zeichenabstand und Serifen. Beispiele hierfür sind Times New Roman und New Century Schoolbook.
FF_SCRIPT	Schriften, die einer Handschrift ähneln, z.B. Script und Cursive.
FF_SWISS	Schriften mit variablem Zeichenabstand und ohne Serifen, z.B. MS Sans Serif.

tmCharSet: Bestimmt den Zeichensatz der Schrift. Die folgenden Werte sind definiert:

Konstante	Wert
ANSI_CHARSET	0
DEFAULT_CHARSET	1
SYMBOL_CHARSET	2
SHIFTJIS_CHARSET	128
OEM_CHARSET	255

tmOverhang: Bestimmt die zusätzliche Breite, die zu einigen synthetisierten Schriften hinzugefügt wird. Wenn Attribute wie fett oder kursiv synthetisiert werden, fügt die grafische Geräteschnittstelle (GDI) oder ein Gerät diese Breite zu einem String für jedes Zeichen und den gesamten String hinzu. Die GDI macht z.B. einen String fett, indem sie den Abstand der Elemente eines Zeichens erweitert und mit einem Versatz überstreicht und macht eine Schrift durch Schrägstellen kursiv. In beiden Fällen ist der String breiter, nachdem das Attribut synthetisiert ist. Bei fetten Strings ist der Überhang die Distanz, mit der das Überstreichen überlagert wird. Bei kursiven Strings ist der Überhang der Abstand, um den der obere Rand der Schrift gegenüber dem unteren Rand schräggestellt wird.

Das Element *tmOverhang* ist für viele kursive und fette TrueType-Schriften Null, weil sie nicht synthetisiert sind. Der Überhang ist z.B. für Courier New Italic Null.

Eine Anwendung, die Rasterschriften verwendet, kann diesen Wert verwenden, um den Abstand zwischen Wörtern mit verschiedenen Attributen zu ermitteln.

tmDigitizedAspectX: Gibt das horizontale Seitenverhältnis des Geräts an, für das die Schrift entworfen wurde.

tmDigitizedAspectY: Gibt das vertikale Seitenverhältnis des Geräts an, für das die Schrift entworfen wurde. Die Verhältnis des Elements *tmDigitizedAspectX* zu *tmDigitizedAspectY* ist das Seitenverhältnis des Geräts, für das die Schrift entworfen wurde.

ntmFlags: Gibt einige Elemente der Schriftart an. Dieses Element kann einen oder mehrere der folgenden Werte annehmen:

NTM_REGULAR
NTM_BOLD
NTM_ITALIC

Die Flags *NTM_BOLD* und *NTM_ITALIC* könnten mit dem ODER-Operator kombiniert werden, um eine fett und kursiv ausgegebene Schrift festzulegen.

ntmSizeEM: Bestimmt die Größe des Messungsquadrats für die Schrift in den Einheiten, für die die Schrift entworfen wurde (in fiktiven Einheiten).

ntmCellHeight: Gibt die Höhe der Schrift in den Einheiten an, für die die Schrift entworfen wurde (in fiktiven Einheiten). Dieser Wert sollte mit dem Wert des Elements *ntmSizeEM* verglichen werden.

ntmAvgWidth: Bestimmt die durchschnittliche Breite von Zeichen in der Schrift in den Einheiten, für die die Schrift entworfen wurde (in fiktiven Einheiten). Dieser Wert sollte mit dem Wert des Elements *ntmSizeEM* verglichen werden.

Hinweis Die Größen in der *NEWTEXTMETRIC*-Struktur werden normalerweise in virtuellen Einheiten angegeben; d.h. sie sind vom aktuellen Abbildungsmodus des Bildschirmkontexts abhängig.

Siehe auch *EnumFontFamilies*, *EnumFonts*, *GetDeviceCaps*, *GetTextMetrics*

NFYLOADSEG 3.1

Definition

```
#include <toolhelp.h>

typedef struct tagNFYLOADSEG {        /* nfyls */
    DWORD dwSize;
    WORD wSelector;
    WORD wSegNum;
    WORD wType;
    WORD wcInstance;
    LPCSTR lpstrModuleName
} NFYLOADSEG;
```

Beschreibung Die *NFYLOADSEG*-Struktur enthält Informationen über das Segment, das geladen wird, wenn der Kernel eine Benachrichtigung zum Laden eines Segments sendet.

Elemente *dwSize:* Gibt die Größe der *NFYLOADSEG*-Struktur in Byte an.

wSelector: Enthält den Selektor des Segments, das geladen wird.

wSegNum: Enthält die Segmentnummer der ausführbaren Datei.

wType: Zeigt den Typ der Informationen im Segment an. Nur das niederwertigste Bit von *wType* wird verwendet. Der Typ kann folgenden Wert haben:

Wert	Bedeutung
0	Das Segment enthält Code.
1	Das Segment enthält Daten.

wcInstance: Bezeichnet die Anwendungsinstanz, die geladen wird.

lpstrModuleName: Zeigt auf einen nullterminierten String, der den Namen des Moduls enthält, zu dem das zu ladende Segment gehört.

Siehe auch *NotifyRegister*

NFYLOGERROR 3.1

Definition

```
#include <toolhelp.h>

typedef struct tagNFYLOGERROR {      /* nfyle        */
    DWORD dwSize;
    UINT wErrCode;
    void FAR* lpInfo;                /* abhängig vom
                                        Fehlerwert   */
} NFYLOGERROR;
```

Beschreibung Die *NFYLOGERROR*-Struktur enthält Informationen über einen Fehler bei der Gültigkeitsprüfung, der den Kernel veranlaßt hat, eine *NFY_LOGERROR*-Benachrichtigung zu senden.

Elemente *dwSize:* Gibt die Größe der *NFYLOGERROR*-Struktur in Byte an.

wErrCode: Bezeichnet den Fehlerwert, der das Senden der Benachrichtigung ausgelöst hat.

lpInfo: Zeigt auf zusätzliche Informationen, die vom Fehlerwert abhängen.

Siehe auch *NotifyRegister*

NFYLOGPARAMERROR 3.1

Definition

```
#include <toolhelp.h>

typedef struct tagNFYLOGPARAMERROR {     /* nfylpe    */
    DWORD dwSize;
    UINT wErrCode;
    FARPROC lpfnErrorAddr;
    void FAR* FAR* lpBadParam;
} NFYLOGPARAMERROR;
```

Beschreibung Die *NFYLOGPARAMERROR*-Struktur enthält Informationen über einen Fehler bei der Gültigkeitsprüfung eines Parameters, der den Kernel veranlaßt hat, eine *NFY_LOGERROR*-Benachrichtigung zu senden.

Elemente *dwSize:* Gibt die Größe der *NFYLOGPARAMERROR*-Struktur in Byte an.

wErrCode: Bezeichnet den Fehlerwert, der das Senden der Benachrichtigung ausgelöst hat.

lpfnErrorAddr: Zeigt auf die Adresse der Funktion mit dem ungültigen Parameter.

lpBadParam: Zeigt auf den Namen des ungültigen Parameters.

Siehe auch *NotifyRegister*

NFYRIP 3.1

Definition

```
#include <toolhelp.h>

typedef struct tagNFYRIP {   /* nfyr */
    DWORD dwSize;
    WORD wIP;
    WORD wCS;
    WORD wSS;
    WORD wBP;
    WORD wExitCode;
} NFYRIP;
```

Beschreibung Die *NFYRIP*-Struktur enthält Informationen über das System, wenn ein System-Debug-Fehler (RIP) auftritt.

Elemente *dwSize:* Gibt die Größe der *NFYRIP*-Struktur in Byte an.

wIP: Enthält den Wert, der zum Zeitpunkt des Fehlers im Register IP enthalten war.

wCS: Enthält den Wert, der zum Zeitpunkt des Fehlers im Register CS enthalten war.

wSS: Enthält den Wert, der zum Zeitpunkt des Fehlers im Register SS enthalten war.

wBP: Enthält den Wert, der zum Zeitpunkt des Fehlers im Register BP enthalten war.

wExitCode: Enthält einen Exit-Code, der den Grund des Fehlers beschreibt.

Hinweis Die Funktion *StackTraceCSIPFirst* verwendet die Werte CS:IP und SS:BP aus dieser Struktur.

Der erste durch diese Werte bezeichnete Frame auf dem Stack zeigt auf die Funktion *FatalExit*.

Der nächste Frame zeigt auf die Routine, die *FatalExit* aufgerufen hat, normalerweise in USER.EXE, GDI.EXE oder KRNL286.EXE bzw. KRNL386.EXE.

Siehe auch *FatalExit*, *NotifyRegister*, *StackTraceCSIPFirst*

NFYSTARTDLL 3.1

Definition

```
#include <toolhelp.h>
typedef struct tagNFYSTARTDLL { /* nfysd */
   DWORD dwSize;
   HMODULE hModule;
   WORD wCS;
   WORD wIP;
} NFYSTARTDLL;
```

Beschreibung Die *NFYSTARTDLL*-Struktur enthält Informationen über die dynamische Linkbibliothek (DLL), die geladen wird, wenn der Kernel eine Benachrichtigung zum Laden einer DLL sendet.

Elemente *dwSize:* Gibt die Größe der *NFYSTARTDLL*-Struktur in Byte an.

hModule: Bezeichnet das Bibliotheksmodul, das geladen wird.

wCS: Enthält den Wert im Register CS zur Zeit des Ladens. Dieser Wert wird mit dem Wert des Elements *wIP* verwendet, um die Ladeadresse der DLL zu bestimmen.

wIP: Enthält den Wert im Register IP zur Zeit des Ladens. Dieser Wert wird mit dem Wert des Elements *wCS* verwendet, um die Ladeadresse der DLL zu bestimmen.

Siehe auch *NotifyRegister*

OLECLIENT 3.1

Definition

```
#include <ole.h>
typedef struct _OLECLIENT { /* oc */
    LPOLECLIENTVTBL lpvtbl;
    .
    .   /* vom Client gelieferte Statusinformationen  */
    .
} OLECLIENT;
```

Beschreibung Die *OLECLIENT*-Struktur zeigt auf eine *OLECLIENTVTBL*-Struktur und kann Statusinformationen speichern, die von der Client-Anwendung verwendet werden.

Elemente *lpvtbl:* Zeigt auf eine Tabelle mit Funktionszeigern für den Client.

Hinweis Server und Objekt-Handler sollten die Statusinformationen der *OLECLIENTVTBL*-Struktur verwenden. Die Verwendung und Bedeutung dieser Informationen hängt vollständig von der Client-Anwendung ab. Weil ein Zeiger auf diese Struktur als Parameter an die Callback-Funktion des Client übergeben wird, ist dies die beste Methode für eine Client-Anwendung, private Statusinformationen über das Objekt zu speichern.

OLECLIENTVTBL 3.1

Definition

```
#include <ole.h>

typedef struct _OLECLIENTVTBL {                    /* ocv */
    int (CALLBACK* CallBack)(LPOLECLIENT,
         OLE_NOTIFICATION, LPOLEOBJECT);
} OLECLIENTVTBL;
```

Beschreibung Die *OLECLIENTVTBL*-Struktur enthält einen Zeiger auf eine Callback-Funktion für die Client-Anwendung.

Hinweis Die als Element *CallBack* übergebene Adresse muß durch die Funktion *MakeProcInstance* erzeugt worden sein.

Callback-Funktion **CallBack**

Deklaration

```
int (FAR PASCAL *CallBack)(lpclient, notification,
  lpobject)

LPOLECLIENT lpclient;
OLE_NOTIFICATION notification;
LPOLEOBJECT lpobject;
```

Beschreibung *CallBack* wird von der Client-Bibliothek über bestimmte Ereignisse in der Server-Anwendung benachrichtigt.

Parameter *lpclient:* Zeigt auf die dem Objekt zugeordnete Client-Struktur. Die Bibliothek ruft diesen Zeiger aus ihrer Objektstruktur ab, wenn eine Benachrichtigung eingeht, verwendet ihn, um die Callback-Funktion zu lokalisieren und übergibt dieser den Zeiger auf die Client-Struktur, damit ihn die Client-Anwendung verwenden kann.

notification: Gibt den Grund für die Benachrichtigung an. Der Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
OLE_CHANGED	Das verknüpfte Objekt ist geändert worden. Diese Benachrichtigung wird nicht für eingebettete Objekte gesendet. Wenn diese Benachrichtung empfangen wird, sollte das Objekt im allgemeinen neugezeichnet oder gespeichert werden.
OLE_CLOSED	Das Objekt ist in seinem Server geschlossen worden. Wenn der Client diese Benachrichtigungsbotschaft erhält, sollte er keine Funktion aufrufen, die eine asynchrone Operation bewirkt, bis er die Steuerung der Programmausführung wiedererlangt hat.

Wert	Bedeutung
OLE_QUERY_PAINT	Eine langwierige Zeichenoperation findet statt. Diese Benachrichtigung erlaubt das Unterbrechen des Zeichnens.
OLE_QUERY_RETRY	Der Server hat auf eine Anforderung geantwortet, daß er beschäftigt ist. Diese Benachrichtigung fordert den Client auf, zu bestimmen, ob die Bibliothek mit der Anforderung fortfahren soll. Wenn die Callback-Funktion *FALSE* zurückgibt, wird die Transaktion mit dem Server nicht fortgesetzt.
OLE_RELEASE	Das Objekt wurde freigegeben, weil eine asynchrone Operation beendet ist. Der Client sollte nicht beenden, bevor alle Objekte freigegeben worden sind. Die Client-Anwendung kann die Funktion *OleQueryReleaseError* aufrufen, um zu bestimmen, ob die Operation erfolgreich war. Sie kann auch mit Hilfe von *OleQueryReleaseMethod* überprüfen, ob die Operation beendet ist.
OLE_RENAMED	Das verknüpfte Objekt ist in seinem Server umbenannt worden. Diese Benachrichtigung dient nur zur Information, weil die Bibliothek ihre Verknüpfungsinformationen automatisch aktualisiert.
OLE_SAVED	Das verknüpfte Objekt ist in seinem Server gespeichert worden. Der Client erhält diese Benachrichtigung, wenn der Server die Funktion *OleSavedServerDoc* als Reaktion auf den Befehl *Aktualisieren* im *Datei*-Menü des Server aufruft.

Wenn der Client die Benachrichtigung *OLE_CLOSED* erhält, speichert er normalerweise die Bedingung und kehrt zur Client-Bibliothek zurück. Er führt nur eine Aktion durch, wenn die Client-Bibliothek die Steuerung der Programmausführung an ihn zurückgibt. Wenn die Client-Anwendung Operationen ausführen muß, bevor sie die Steuerung wiedererlangt hat, sollte sie keine Funktionen aufrufen, die eine asynchrone Operation auslösen könnten.

lpobject: Zeigt auf das Objekt, das das Senden der Benachrichtigung verursacht hat. Anwendungen, die die gleiche Client-Struktur für mehr als ein Objekt verwenden, können durch den Parameter *lpobject* die Benachrichtigung dem richtigen Objekt zuordnen.

Rückgabewert Wenn der Parameter *notification* entweder *OLE_QUERY_PAINT* oder *OLE_QUERY_RETRY* angibt, sollte der Client *TRUE* zurückgeben, wenn die Bibliothek fortfahren soll oder *FALSE*, um die Zeichenoperation oder die Server-Transaktion abzubrechen. Wenn der Parameter *notification* weder *OLE_QUERY_PAINT* noch *OLE_QUERY_RETRY* angibt, wird der Rückgabewert ignoriert.

Hinweis Die Client-Anwendung sollte auf diese Benachrichtigungen so schnell wie möglich reagieren, z.B. im Rahmen der Hauptereignisschleife oder wenn sie das Objekt schließt. Die Aktualisierung eines Objekts kann solange verzögert werden, bis der Benutzer die Aktualisierung anfordert, wenn der Client dies unterstützt. Der Client kann die Bibliothek auch aus dieser Benachrichtigungs-Callback-Funktion aufrufen, da die Bibliothek wiedereintretend ist. Der Client sollte keine asynchrone Operation ausführen, während bestimmte andere Operationen durchgeführt werden (z.B. das Öffnen oder Löschen eines Objekts). Er sollte auch in keine Botschaftsschleife innerhalb der Callback-Funktion eintreten. Wenn die Client-Anwendung eine Funktion aufruft, die eine asynchrone Operation verursachen würde, liefert die Client-Bibliothek *OLE_WAIT_FOR_RELEASE* zurück. Sie benachrichtigt in diesem Fall die Anwendung, sobald die Operation beendet ist, indem sie *OLE_RELEASE* verwendet und gibt *OLE_BUSY* zurück, wenn der Client eine konfligierende Operation einleitet, während die vorhergehende noch im Gange ist. Der Client kann ermitteln, ob eine asynchrone Operation stattfindet, indem er die Funktion *OleQueryReleaseStatus* aufruft. Diese Funktion gibt *OLE_BUSY* zurück, wenn die Operation noch nicht beendet ist.

Siehe auch *OleQueryReleaseStatus*

OLEOBJECT 3.1

Definition

```
#include <ole.h>

typedef struct _OLEOBJECT { /* oo */
    LPOLEOBJECTVTBL lpvtbl;
    .
    .    /* vom Server gelieferte Statusinformationen */
    .
} OLEOBJECT;
```

Beschreibung Die *OLEOBJECT*-Struktur zeigt auf eine Tabelle von Funktionszeigern für ein Objekt. Diese Struktur wird von Server-Anwendungen für die Server-Bibliothek initialisiert und verwaltet.

Elemente *lpvtbl:* Zeigt auf eine Tabelle von Funktionszeigern für das Objekt.

OLEOBJECTVTBL 3.1

Definition

```
#include <ole.h>

typedef struct _OLEOBJECTVTBL { /* oov */
   void FAR* (CALLBACK* QueryProtocol)(LPOLEOBJECT,
       OLE_LPCSTR);
   OLESTATUS (CALLBACK* Release)(LPOLEOBJECT);
   OLESTATUS (CALLBACK* Show)(LPOLEOBJECT, BOOL);
   OLESTATUS (CALLBACK* DoVerb)(LPOLEOBJECT, UINT,
       BOOL, BOOL);
   OLESTATUS (CALLBACK* GetData)(LPOLEOBJECT,
       OLECLIPFORMAT, HANDLE FAR*);
   OLESTATUS (CALLBACK* SetData)(LPOLEOBJECT,
       OLECLIPFORMAT, HANDLE);
   OLESTATUS (CALLBACK* SetTargetDevice)(LPOLEOBJECT,
       HGLOBAL);
   OLESTATUS (CALLBACK* SetBounds)(LPOLEOBJECT,
       OLE_CONST RECT FAR*);
   OLECLIPFORMAT (CALLBACK* EnumFormats)(LPOLEOBJECT,
       OLECLIPFORMAT);
   OLESTATUS (CALLBACK* SetColorScheme)(LPOLEOBJECT,
       OLE_CONST LOGPALETTE FAR*);

    /* Server-Anwendungen implementieren nur die oben
     * aufgelisteten Funktionen. Objekt-Handler können
     * die Funktionen in dieser Strukur verwenden, um
     * die voreingestellte Verhaltensweise eines Server
     * zu ändern.*/

   OLESTATUS (CALLBACK* Delete)(LPOLEOBJECT);
   OLESTATUS (CALLBACK* SetHostNames)(LPOLEOBJECT,
       OLE_LPCSTR, OLE_LPCSTR);
   OLESTATUS (CALLBACK* SaveToStream)(LPOLEOBJECT,
       LPOLESTREAM);
   OLESTATUS (CALLBACK* Clone)(LPOLEOBJECT,
       LPOLECLIENT, LHCLIENTDOC, OLE_LPCSTR,
       LPOLEOBJECT FAR*);
   OLESTATUS (CALLBACK* CopyFromLink)(LPOLEOBJECT,
       LPOLECLIENT, LHCLIENTDOC, OLE_LPCSTR,
       LPOLEOBJECT FAR*);
   OLESTATUS (CALLBACK* Equal)(LPOLEOBJECT,
       LPOLEOBJECT);
   OLESTATUS (CALLBACK* CopyToClipboard)(LPOLEOBJECT);
   OLESTATUS (CALLBACK* Draw)(LPOLEOBJECT, HDC,
       OLE_CONST RECT FAR*, OLE_CONST RECT FAR*, HDC);
   OLESTATUS (CALLBACK* Activate)(LPOLEOBJECT, UINT,
       BOOL, BOOL, HWND, OLE_CONST RECT FAR*);
   OLESTATUS (CALLBACK* Execute)(LPOLEOBJECT, HGLOBAL,
       UINT);
```

```
    OLESTATUS (CALLBACK* Close)(LPOLEOBJECT);
    OLESTATUS (CALLBACK* Update)(LPOLEOBJECT);
    OLESTATUS (CALLBACK* Reconnect)(LPOLEOBJECT);
    OLESTATUS (CALLBACK* ObjectConvert)(LPOLEOBJECT,
        OLE_LPCSTR, LPOLECLIENT, LHCLIENTDOC,
    OLE_LPCSTR, LPOLEOBJECT FAR*);
    OLESTATUS (CALLBACK* GetLinkUpdateOptions)
        (LPOLEOBJECT, OLEOPT_UPDATE FAR*);
    OLESTATUS (CALLBACK* SetLinkUpdateOptions)
        (LPOLEOBJECT, OLEOPT_UPDATE);
    OLESTATUS (CALLBACK* Rename)(LPOLEOBJECT,OLE_LPCSTR);
    OLESTATUS (CALLBACK* QueryName)(LPOLEOBJECT, LPSTR,
        UINT FAR*);
    OLESTATUS (CALLBACK* QueryType)(LPOLEOBJECT,
        LONG FAR*);
    OLESTATUS (CALLBACK* QueryBounds)(LPOLEOBJECT,
        RECT FAR*);
    OLESTATUS (CALLBACK* QuerySize)(LPOLEOBJECT,
        DWORD FAR*);
    OLESTATUS (CALLBACK* QueryOpen)(LPOLEOBJECT);
    OLESTATUS (CALLBACK* QueryOutOfDate)(LPOLEOBJECT);
    OLESTATUS (CALLBACK* QueryReleaseStatus)
        (LPOLEOBJECT);
    OLESTATUS (CALLBACK* QueryReleaseError)(LPOLEOBJECT);
    OLE_RELEASE_METHOD (CALLBACK* QueryReleaseMethod)
        (LPOLEOBJECT);
    OLESTATUS (CALLBACK* RequestData)(LPOLEOBJECT,
        OLECLIPFORMAT);
    OLESTATUS (CALLBACK* ObjectLong)(LPOLEOBJECT, UINT,
        LONG FAR*);
} OLEOBJECTVTBL;
```

Beschreibung

Die Elemente der *OLEOBJECTVTBL*-Struktur zeigen auf Funktionen, die ein Objekt manipulieren. Eine Server-Anwendung erzeugt sie zusammen mit einer *OLEOBJECT*-Struktur, um der Server-Bibliothek den Zugriff auf ein Objekt zu ermöglichen.

Server-Anwendungen müssen nur die Funktion *SetColorScheme* implementieren. Objekt-Handler können spezielle Versionen für einige oder alle Funktionen in dieser Struktur implementieren.

Hinweis

Folgende Funktionen in *OLEOBJECTVTBL* sollten *OLE_BUSY* zurückgeben, wenn dies dem aktuellen Status entspricht:

Activate	*Close*	*CopyFromLink*
Delete	*DoVerb*	*Execute*
ObjectConvert	*Reconnect*	*RequestData*
SetBounds	*SetColorScheme*	*SetData*
SetHostNames	*SetLinkUpdateOptions*	*SetTargetDevice*
Show	*Update*	

Im folgenden werden in alphabetischer Reihenfolge einige der Callback-Funktionen beschrieben, auf die die Elemente der *OLEOBJECTVTBL*-Struktur zeigen. Informationen über die nicht aufgeführten Funktionen finden Sie in der Dokumentation der entsprechenden OLE-Funktion. Benötigen Sie z.B. weitere Informationen über das Element *QueryProtocol*, sehen Sie unter *OleQueryProtocol* nach.

Callback-Funktion **DoVerb**

Syntax

```
OLESTATUS (FAR PASCAL *DoVerb)(lpObject, iVerb, fShow,
    fTakeFocus)

LPOLEOBJECT lpObject;
UINT iVerb;
BOOL fShow;
BOOL fTakeFocus;
```

Beschreibung Die Funktion *DoVerb* bestimmt, welche Art von Aktion der Server ausführen soll, wenn der Benutzer ein Objekt aktiviert.

Parameter *lpObject:* Zeigt auf das Objekt, das aktiviert werden soll.

iVerb: Legt die Art der durchzuführenden Aktion fest. Der Wert dieses Parameters wird von der Server-Anwendung bestimmt.

fShow: Gibt an, ob das Server-Fenster gezeigt werden soll. Dieser Wert ist *TRUE*, wenn das Fenster gezeigt werden soll, andernfalls *FALSE*.

fTakeFocus: Bestimmt, ob das Server-Fenster den Fokus erhält. Wenn das Server-Fenster den Fokus erhalten soll, ist dieser Wert *TRUE*, andernfalls *FALSE*. Dieser Parameter ist nur von Bedeutung, wenn der Parameter *fShow TRUE* ist.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis Alle Server müssen das Bearbeiten von Objekten unterstützen. Wenn ein Server keine Aktionsart außer »Edit« unterstützt, sollte er das Objekt bearbeiten, gleichgültig welcher Wert im Parameter *iVerb* angegeben wird.

Callback-Funktion **GetData**

Deklaration

```
OLESTATUS (FAR PASCAL *GetData)(lpObject, cfFormat,
   lphdata)

LPOLEOBJECT lpObject;
OLECLIPFORMAT cfFormat;
HANDLE FAR* lphdata;
```

Beschreibung Die Funktion *GetData* ruft Daten von einem Objekt in einem bestimmten Format ab. Die Server-Anwendung sollte Speicher reservieren, ihn mit den Daten füllen und die Daten über den Parameter *lphdata* zurückgeben.

Parameter *lpObject:* Zeigt auf die *OLEOBJECT*-Struktur, von der Daten angefordert werden.

cfFormat: Legt das Format fest, in dem die Daten angefordert werden.

lphdata: Zeigt auf das Handle des reservierten Speichers, das die Server-Anwendung zurückgibt. Die Bibliothek gibt den Speicher frei, wenn er nicht mehr benötigt wird.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls kann einer der folgenden Fehlerwerte zurückgegeben werden:

OLE_ERROR_BLANK
OLE_ERROR_FORMAT
OLE_ERROR_OBJECT

Callback-Funktion **ObjectLong**

Syntax
```
OLESTATUS (FAR PASCAL *ObjectLong)(lpObject, wFlags,
    lpData)

LPOLEOBJECT lpObject;
UINT wFlags;
LONG FAR* lpData;
```

Beschreibung Die Funktion *ObjectLong* erlaubt der aufrufenden Anwendung, Daten zusammen mit einem Objekt zu speichern. Objekt-Handler benutzen normalerweise diese Funktion.

Parameter *lpObject:* Zeigt auf die *OLEOBJECT*-Struktur, für die die Daten gespeichert werden.

wFlags: Bestimmt die Methode für das Setzen und Anfordern von Daten. Dieses Element kann einen oder mehrere der folgenden Werte annehmen:

Wert	Bedeutung
OF_SET	Die Daten werden an die durch den Parameter *lpData* festgelegte Position geschrieben und überschreiben alle bereits vorhandenen Daten.
OF_GET	Die Daten werden von der durch den Parameter *lpData* angegebenen Position gelesen.

Wert	Bedeutung
OF_HANDLER	Die Daten werden von einem Objekt-Handler bearbeitet, geschrieben oder gelesen. Dieser Wert verhindert, daß Daten von einem Objekt-Handler von anderen Anwendungen überschrieben werden.

Wenn die aufrufende Anwendung *OF_SET* oder *OF_GET* angibt, liefert die Funktion einen Zeiger auf die vorhergehenden Daten zurück. Die durch *lpData* bezeichneten Daten werden durch die Daten der aufrufenden Anwendung ersetzt.

lpData: Zeigt auf die Daten, die gelesen oder geschrieben werden sollen.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Callback-Funktion **Release**

Syntax
```
OLESTATUS (FAR PASCAL *Release)(lpObject)

LPOLEOBJECT lpObject;
```

Beschreibung Die Funktion *Release* veranlaßt den Server, die Ressourcen freizugeben, die der angegebenen *OLEOBJECT*-Struktur zugeordnet sind.

Parameter *lpObject:* Zeigt auf die *OLEOBJECT*-Struktur, die freigegeben werden soll.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis Die Server-Anwendung sollte keine Daten freigeben und entfernen, wenn die Bibliothek die Funktion *Release* aufruft. Die Bibliothek ruft diese Funktion nur auf, wenn keine Client-Anwendungen mit dem Objekt verbunden sind.

Callback-Funktion **Show**

Syntax
```
OLESTATUS (FAR PASCAL *Show)(lpObject, fTakeFocus)

LPOLEOBJECT lpObject;
BOOL fTakeFocus;
```

Beschreibung Die Funktion *Show* veranlaßt den Server, ein Objekt zu zeigen, indem dessen Fenster dargestellt und nötigenfalls ein Bildlauf durchgeführt wird.

Parameter *lpObject:* Zeigt auf die *OLEOBJECT*-Struktur, die gezeigt werden soll.

fTakeFocus: Bestimmt, ob das Server-Fenster den Fokus erhält. Wenn das Server-Fenster den Fokus erhalten soll, ist dieser Wert *TRUE*, andernfalls *FALSE*.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis Die Bibliothek ruft die Funktion *Show* auf, wenn die Server-Anwendung dem Benutzer das Dokument zum Bearbeiten anzeigen soll oder um den Server aufzufordern, einen Bildlauf im Dokument durchzuführen, um das Objekt sichtbar zu machen.

Callback-Funktion **SetColorScheme**

Deklaration

```
OLESTATUS SetColorScheme(lpObject, lpPal)

LPOLEOBJECT lpObject;
OLE_CONST LOGPALETTE FAR* lpPal
```

Beschreibung Die Funktion *SetColorScheme* sendet die Farbpalette an den Server, die von der Client-Anwendung empfohlen wurde.

Parameter *lpObject:* Zeigt auf eine *OLEOBJECT*-Struktur, für die die Client-Anwendung eine Palette empfiehlt.

lpPal: Zeigt auf eine *LOGPALETTE*-Struktur, die die empfohlene Palette angibt.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis Server-Anwendungen müssen die von der Client-Anwendung empfohlene Palette nicht verwenden.

Bevor sie aus der Funktion *SetColorScheme* zurückkehrt, sollte die Server-Anwendung die durch den Parameter *lpPal* bezeichnete Palette in einem Aufruf der Funktion *CreatePalette* verwenden, um ein Handle für die Palette zu erzeugen:

```
hpal = CreatePalette(lpPal);
```

Der Server kann das Handle verwenden, um sich auf die Palette zu beziehen.

Der erste Paletteneintrag in der *LOGPALETTE*-Struktur legt die von der Client-Anwendung empfohlene Vordergrundfarbe fest. Der zweite bestimmt die Hintergrundfarbe. Die erste Hälfte der restlichen Paletteneinträge sind Füllfarben und die zweite Hälfte Farben für Linien und Text.

Client-Anwendungen geben normalerweise eine gerade Anzahl von Paletteneinträgen an. Wenn es eine ungerade Anzahl gibt, sollte der Server den ungeraden Eintrag als Füllfarbe verwenden. Wenn es z.B. fünf Einträge gibt, sollten drei als Füllfarben und zwei als Farben für Linien und Text interpretiert werden.

Callback-Funktion **SetData**

Syntax

```
OLESTATUS (FAR PASCAL *SetData)(lpObject, cfFormat,
   hdata)

LPOLEOBJECT lpObject;
OLECLIPFORMAT cfFormat;
HANDLE hdata;
```

Beschreibung *SetData* speichert Daten in einem Objekt im angegebenen Format ab. Sie wird mit dem Native-Datenformat aufgerufen, wenn ein Client ein eingebettetes Objekt zum Bearbeiten öffnet. Diese Funktion wird auch verwendet, wenn der Client die Funktion *OleSetData* mit einem anderen Format aufruft.

Parameter *lpObject:* Zeigt auf die *OLEOBJECT*-Struktur, in der die Daten gespeichert sind.

cfFormat: Gibt das Datenformat an.

hdata: Bezeichnet eine Speicherposition, von der die Server-Anwendung die Daten einlesen soll. Der Server sollte dieses Handle löschen, nachdem er die Daten bearbeitet hat.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis Die Server-Anwendung ist für den Speicher verantwortlich, der vom Parameter *hdata* bezeichnet wird. Sie muß diese Daten löschen, auch wenn sie *OLE_BUSY* zurückgibt oder ein Fehler auftritt.

Callback-Funktion **SetTargetDevice**

Syntax

```
OLESTATUS (FAR PASCAL *SetTargetDevice)(lpObject, hotd)

LPOLEOBJECT lpObject;
HGLOBAL hotd;
```

Beschreibung Die Funktion *SetTargetDevice* liefert Informationen über das Zielgerät des Client für das Objekt. Der Server kann diese Informationen verwenden, um Ausgaben dem Zielgerät anzupassen.

Parameter *lpObject:* Zeigt auf die *OLEOBJECT*-Struktur, für die das Zielgerät angegeben ist.

hotd: Bezeichnet eine *OLETARGETDEVICE*-Struktur.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis Die Server-Anwendung ist für den Speicher verantwortlich, der vom Parameter *hdata* bezeichnet wird. Sie muß diese Daten

löschen, auch wenn sie *OLE_BUSY* zurückgibt oder ein Fehler auftritt.

Die Bibliothek übergibt *NULL* für den Parameter *hdata*, um anzuzeigen, daß die Ausgabe auf dem Bildschirm erfolgen soll.

Siehe auch *OleSetTargetDevice*

OLESERVER 3.1

Definition

```
#include <ole.h>

typedef struct _OLESERVER { /* os */
    LPOLESERVERVTBL lpvtbl;
    .
    .    /* vom Server gelieferte Statusinformationen */
    .
} OLESERVER;
```

Beschreibung Die *OLESERVER*-Struktur zeigt auf eine Tabelle von Funktionszeigern für den Server. Diese Tabelle wird von Server-Anwendungen für die Server-Bibliothek initialisiert und verwaltet.

Elemente *lpvtbl:* Zeigt auf eine Tabelle von Funktionszeigern für den Server.

OLESERVERDOC 3.1

Definition

```
#include <ole.h>

typedef struct _OLESERVERDOC { /* osd */
    LPOLESERVERDOCVTBL lpvtbl;
    .
    .  /* vom Server gelieferte Statusinformationen */
    .
} OLESERVERDOC;
```

Beschreibung Die *OLESERVERDOC*-Struktur zeigt auf eine Tabelle von Funktionszeigern für ein Dokument. Diese Tabelle wird von Server-Anwendungen für die Server-Bibliothek initialisiert und verwaltet.

Elemente *lpvtbl:* Zeigt auf eine Tabelle von Funktionszeigern für das Dokument.

OLESERVERDOCVTBL 3.1

Definition

```
#include <ole.h>

typedef struct _OLESERVERDOCVTBL { /* odv */
    OLESTATUS (CALLBACK* Save)(LPOLESERVERDOC);
    OLESTATUS (CALLBACK* Close)(LPOLESERVERDOC);
```

```
    OLESTATUS (CALLBACK* SetHostNames)(LPOLESERVERDOC,
        OLE_LPCSTR, OLE_LPCSTR);
    OLESTATUS (CALLBACK* SetDocDimensions)
        (LPOLESERVERDOC, OLE_CONST RECT FAR*);
    OLESTATUS (CALLBACK* GetObject)(LPOLESERVERDOC,
        OLE_LPCSTR, LPOLEOBJECT FAR*, LPOLECLIENT);
    OLESTATUS (CALLBACK* Release)(LPOLESERVERDOC);
    OLESTATUS (CALLBACK* SetColorScheme)
        (LPOLESERVERDOC, OLE_CONST LOGPALETTE FAR*);
    OLESTATUS (CALLBACK* Execute)(LPOLESERVERDOC,
        HGLOBAL);
} OLESERVERDOCVTBL;
```

Beschreibung Die *OLESERVERDOCVTBL*-Struktur zeigt auf Funktionen, die ein Dokument manipulieren. Eine Server-Anwendung erzeugt sie zusammen mit einer *OLESERVERDOC*-Struktur, um der Server-Bibliothek den Zugriff auf ein Dokument zu ermöglichen. Dokumente, die auf Anforderung der Bibliothek geöffnet oder erzeugt worden sind, sollten dem Benutzer nicht zum Bearbeiten gezeigt werden, bis die Bibliothek dies anfordert.

Jede Funktion außer *Release* kann *OLE_BUSY* zurückgeben.

Im folgenden werden in alphabetischer Reihenfolge alle Callback-Funktionen beschrieben, auf die die Elemente der *OLESERVERDOCVTBL*-Struktur zeigen.

Callback-Funktion **Close**

Syntax

```
OLESTATUS (FAR PASCAL *Close)(lpDoc)

LPOLESERVERDOC lpDoc;
```

Beschreibung Die Funktion *Close* weist die Server-Anwendung an, das Dokument zu schließen. Die Bibliothek ruft diese Funktion auf, wenn die Client-Anwendung das Schließen einleitet.

Parameter *lpDoc:* Zeigt auf eine *OLESERVERDOC*-Struktur, die dem zu schließenden Dokument entspricht.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis Die Bibliothek ruft die Funktion *Close* immer vor der Funktion *Release* der *OLESERVERVTBL*-Struktur auf.

Die Server-Anwendung sollte den Benutzer nicht auffordern, das Dokument zu speichern oder andere Aktionen durchzuführen. Meldungen dieser Art werden von der Client-Anwendung durchgeführt.

Wenn die Bibliothek die Funktion *Close* aufruft, sollte der Server durch den Aufruf von *OleRevokeServerDoc* reagieren. Die Ressourcen für das Dokument werden freigegeben, wenn die Bibliothek die Funktion *Release* aufruft. Der Server sollte nicht

auf diese Funktion warten, indem er nach dem Aufruf von *OleRevokeServerDoc* in eine Botschaftsschleife eintritt. Ein Server sollte niemals in eine Botschaftsschleife eintreten, während er eine dieser Funktionen bearbeitet.

Wenn ein Dokument geschlossen ist, sollte der Server den Speicher für die *OLESERVERDOCVTBL*-Struktur und die zugeordneten Ressourcen freigeben.

Callback-Funktion **Execute**

Syntax

```
OLESTATUS (FAR PASCAL *Execute)(lpDoc, hCommands)

LPOLESERVERDOC lpDoc;
HGLOBAL hCommands;
```

Beschreibung Die Funktion *Execute* erhält *WM_DDE_EXECUTE*-Befehle, die von Client-Anwendungen durch einen Aufruf der Funktion *OleExecute* gesendet werden.

Parameter lpDoc: Zeigt auf eine *OLESERVERDOC*-Struktur, auf die die Befehle des dynamischen Datenaustauschs angewendet werden.

hCommands: Bezeichnet Speicher, der einen oder mehrere auszuführende DDE-Befehle enthält.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis Der Server sollte das im Parameter *hCommands* angegebene Handle niemals freigeben.

Callback-Funktion **GetObject**

Syntax

```
OLESTATUS (FAR PASCAL *GetObject)(lpDoc, lpszItem,
  lplpObject, lpClient)

LPOLESERVERDOC lpDoc;
OLE_LPCSTR lpszItem;
LPOLEOBJECT FAR* lplpObject;
LPOLECLIENT lpClient;
```

Beschreibung Die Funktion *GetObject* fordert den Server auf, eine *OLEOBJECT*-Struktur zu erzeugen.

Parameter *lpDoc:* Zeigt auf eine *OLESERVERDOC*-Struktur, die diesem Dokument entspricht.

lpszItem: Zeigt auf einen nullterminierten String, der den Namen eines Elements im angegebenen Dokument festlegt, für das eine Objektstruktur angefordert wird. Wenn dieser String auf *NULL* gesetzt ist, wird das gesamte Dokument angefordert. Dieser String darf keine Schrägstriche (/) enthalten.

lplpObject: Zeigt auf eine Variable vom Typ *LPOLEOBJECT*, in der die Server-Anwendung einen **long-**Zeiger auf die reservierte *OLEOBJECT*-Struktur zurückgeben sollte.

lpClient: Zeigt auf eine von der Bibliothek reservierte *OLECLIENT*-Struktur. Der Server sollte diese Struktur dem Objekt zuordnen und es verwenden, um die Bibliothek von Änderungen des Objekts zu benachrichtigen.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis Die Server-Anwendung sollte die *OLEOBJECT*-Struktur reservieren und initialisieren, sie der *OLECLIENT*-Struktur zuordnen, auf die der Parameter *lpClient* zeigt und einen Zeiger auf die *OLEOBJECT*-Struktur über den Parameter *lplpObject* zurückliefern. Die Bibliothek ruft die Funktion *GetObject* auf, um einem Client den durch den Parameter *lpszItem* bezeichneten Teil eines Dokuments zuzuordnen. Nachdem die Zuordnung von Client und Objekt erfolgt ist, kann der Server Benachrichtigungen an den Client senden.

Anwendungen sollten darauf vorbereitet sein, mehrere Aufrufe von *GetObject* für ein gegebenes Objekt zu bearbeiten. Dies erfordert das Erzeugen mehrerer *OLECLIENT*-Strukturen und das Senden von Benachrichtigungen an jede dieser Strukturen. Mehrere Aufrufe von *GetObject* sind möglich, weil einige Client-Anwendungen das Verknüpfen und Einbetten von Objekten (OLE) implementieren, indem sie den dynamischen Datenaustausch (DDE) und nicht die dynamischen OLE-Linkbibliotheken verwenden. Diese Anwendungen können sowohl *NULL* als auch den tatsächlichen Elementnamen für den Parameter *lpszItem* verwenden.

Callback-Funktion **Release**

Syntax

```
OLESTATUS (FAR PASCAL *Release)(lpDoc)

LPOLESERVERDOC lpDoc;
```

Beschreibung Die Funktion *Release* benachrichtigt den Server, wenn ein Dokument, für das der Zugriff aufgehoben wurde, Konversationen beendet hat und freigegeben und entfernt werden kann.

Parameter *lpDoc:* Zeigt auf eine *OLESERVERDOC*-Struktur, für deren Handle der Zugriff aufgehoben wurde und jetzt freigegeben werden kann.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Callback-Funktion **Save**

Syntax
```
OLESTATUS (FAR PASCAL *Save)(lpDoc)

LPOLESERVERDOC lpDoc;
```

Beschreibung Die Funktion *Save* weist den Server an, das Dokument zu speichern.

Parameter *lpDoc:* Zeigt auf eine *OLESERVERDOC*-Struktur, die dem zu speichernden Dokument entspricht.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Callback-Funktion **SetColorScheme**

Syntax
```
OLESTATUS (FAR PASCAL *SetColorScheme)(lpDoc, lpPal)

LPOLESERVERDOC lpDoc;
OLE_CONST LOGPALETTE FAR* lpPal;
```

Beschreibung Die Funktion *SetColorScheme* sendet die Farbpalette an den Server, die von der Client-Anwendung empfohlen wurde.

Parameter *lpObject:* Zeigt auf eine *OLEOBJECT*-Struktur, für die die Client-Anwendung eine Palette empfiehlt.

lpPal: Zeigt auf eine *LOGPALETTE*-Struktur, die die empfohlene Palette angibt.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis Es ist nicht erforderlich, daß Server-Anwendungen die von der Client-Anwendung empfohlene Palette verwenden.

Bevor sie aus der Funktion *SetColorScheme* zurückspringt, sollte die Server-Anwendung ein Handle für die Palette erzeugen. Dazu sollt sie die Palette, auf die der Parameter *lpPal* zeigt, in einem Aufruf der Funktion *CreatePalette* verwenden, wie im folgenden Beispiel gezeigt:

```
hpal = CreatePalette(lpPal);
```

Der Server kann das Handle verwenden, um sich auf die Palette zu beziehen.

Der erste Paletteneintrag in der *LOGPALETTE*-Struktur legt die von der Client-Anwendung empfohlene Vordergrundfarbe fest. Der zweite bestimmt die Hintergrundfarbe. Die erste Hälfte der restlichen Paletteneinträge sind Füllfarben und die zweite Hälfte Farben für Linien und Text.

Client-Anwendungen geben normalerweise eine gerade Anzahl von Paletteneinträgen an. Gibt es eine ungerade Anzahl, sollte der Server den ungeraden Eintrag als Füllfarbe verwen-

den. Wenn es zum Beispiel fünf Einträge gibt, sollten drei als Füllfarben und zwei als Farben für Linien und Text interpretiert werden.

Callback-Funktion **SetDocDimensions**

Syntax
```
OLESTATUS (FAR PASCAL *SetDocDimensions)(lpDoc, lpRect)

LPOLESERVERDOC lpDoc;
OLE_CONST RECT FAR* lpRect;
```

Beschreibung Die Funktion *SetDocDimensions* übergibt dem Server das Rechteck auf dem Zielgerät, für das das Objekt formatiert werden soll. Diese Funktion ist nur für Dokumente von Bedeutung, die eingebettete Objekte sind.

Parameter *lpDoc:* Zeigt auf die *OLESERVERDOC*-Struktur, die einem Dokument entspricht. Dieses Dokument ist das eingebettete Objekt, für das ein Zielgerät angegeben wird.

lpRect: Zeigt auf eine *RECT*-Struktur, die die Zielgröße des Objekts in *MM_HIMETRIC*-Einheiten enthält. Im Abbildungsmodus *MM_HIMETRIC* verläuft die positive y-Richtung nach oben.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Callback-Funktion **SetHostNames**

Syntax
```
OLESTATUS (FAR PASCAL *SetHostNames)(lpDoc, lpszClient,
  lpszDoc)

LPOLESERVERDOC lpDoc;
OLE_LPCSTR lpszClient;
OLE_LPCSTR lpszDoc;
```

Beschreibung *SetHostNames* legt den Namen für einen Fenstertitel fest. Dieser wird nur für ein eingebettetes Objekt verwendet, da ein verknüpftes Objekt seinen eigenen Titel hat. Diese Funktion wird nur für Dokumente verwendet, die eingebettete Objekte sind.

Parameter *lpDoc:* Zeigt auf eine *OLESERVERDOC*-Struktur, die einem Dokument entspricht. Dieses Dokument stellt das eingebettete Objekt dar, für das ein Name festgelegt wird.

lpszClient: Zeigt auf einen nullterminierten String, der den Namen des Client angibt.

lpszDoc: Zeigt auf einen nullterminierten String, der den Namen des Client für das Objekt angibt.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

OLESERVERVTBL 3.1

Definition

```
#include <ole.h>

typedef struct _OLESERVERVTBL {                /* osv */
    OLESTATUS (CALLBACK* Open)(LPOLESERVER, LHSERVERDOC,
        OLE_LPCSTR, LPOLESERVERDOC FAR*);
    OLESTATUS (CALLBACK* Create)(LPOLESERVER,
        LHSERVERDOC, OLE_LPCSTR, OLE_LPCSTR,
        LPOLESERVERDOC FAR*);
    OLESTATUS (CALLBACK* CreateFromTemplate)(LPOLESERVER,
        LHSERVERDOC, OLE_LPCSTR, OLE_LPCSTR, OLE_LPCSTR
        LPOLESERVERDOC FAR*);
    OLESTATUS (CALLBACK* Edit)(LPOLESERVER, LHSERVERDOC,
        OLE_LPCSTR, OLE_LPCSTR, LPOLESERVERDOC FAR*);
    OLESTATUS (CALLBACK* Exit)(LPOLESERVER);
    OLESTATUS (CALLBACK* Release)(LPOLESERVER);
    OLESTATUS (CALLBACK* Execute)(LPOLESERVER);
} OLESERVERVTBL;
```

Beschreibung

Elemente der *OLESERVERVTBL*-Struktur zeigen auf Funktionen, die einen Server manipulieren. Nachdem eine Server-Anwendung diese Struktur und eine *OLESERVER*-Struktur erzeugt hat, kann die Server-Bibliothek Operationen mit dem Server durchführen.

Jede dieser Callback-Funktionen außer *Release* kann *OLE_BUSY* zurückgeben.

Im folgenden werden in alphabetischer Reihenfolge alle Callback-Funktionen beschrieben, auf die die Elemente der *OLESERVERVTBL*-Struktur zeigen.

Callback-Funktion

Create

Deklaration

```
OLESTATUS (FAR PASCAL *Create)(lpServer, lhDoc,
  lpszClass, lpszDoc, lplpDoc);

LPOLESERVER lpServer;        /* Handle für den Server */
LHSERVERDOC lhDoc;           /* Handle für das
                                Dokument              */
OLE_LPCSTR lpszClass;        /* Name der Klasse       */
OLE_LPCSTR lpszDoc;          /* Name des Dokuments    */
LPOLESERVERDOC FAR* lplpDoc; /* Platzhalter für die
                                zurückgegebene
                                OLESERVERDOC-Struktur */
```

Beschreibung

Die Funktion *Create* erzeugt ein neues Objekt, das in die Client-Anwendung eingebettet werden soll. Der Parameter *lpszDoc* bezeichnet das Objekt, sollte aber nicht zum Erzeugen einer Datei für das Objekt verwendet werden.

Parameter

lpServer: Zeigt auf eine *OLESERVER*-Struktur, die den Server bezeichnet.

lhDoc: Bezeichnet das Dokument. Die Bibliothek verwendet dieses Handle intern.

lpszClass: Zeigt auf einen nullterminierten String, der die Klasse des zu erzeugenden Dokuments angibt.

lpszDoc: Zeigt auf einen nullterminierten String, der den Namen des Dokuments enthält, das erzeugt werden soll. Dieser Name kann verwendet werden, um das Dokument in Fenstertiteln zu bezeichnen.

lplpDoc: Zeigt auf eine Variable vom Typ *LPOLESERVERDOC*, in der die Server-Anwendung einen **long**-Zeiger auf die erzeugte *OLESERVERDOC*-Struktur zurückgeben sollte.

Rückgabewert

Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis

Wenn die Bibliothek diese Funktion aufruft, erzeugt die Server-Anwendung ein Dokument einer bestimmten Klasse, reserviert und initialisiert eine *OLESERVERDOC*-Struktur, ordnet das Handle der Bibliothek diesem Dokument zu und liefert die Adresse der Struktur zurück. Diese Funktion öffnet das erzeugte Dokument zum Bearbeiten und bettet es in die Client-Anwendung ein, wenn es aktualisiert oder geschlossen wird.

Server-Anwendungen verfolgen oft die im Dokument vorgenommenen Änderungen, so daß der Benutzer aufgefordert werden kann, die Änderungen zu speichern, falls dies notwendig ist.

Callback-Funktion

CreateFromTemplate

Deklaration

```
OLESTATUS (FAR PASCAL *CreateFromTemplate)(lpServer,
   hDoc, lpszClass, lpszDoc, lpszTemplate, lplpDoc);

LPOLESERVER lpServer;        /* Handle für den Server */
LHSERVERDOC lhDoc;           /* Handle für das
                                Dokument             */
OLE_LPCSTR lpszClass;        /* Name der Klasse      */
OLE_LPCSTR lpszDoc;          /* Name des Dokuments   */
OLE_LPCSTR lpszTemplate;     /* Name der Schablone
                                für das Dokument     */
LPOLESERVERDOC FAR* lplpDoc; /* Platzhalter für die
                                zurückgegebene
                                OLESERVERDOC-Struktur */
```

Beschreibung

Die Funktion *CreateFromTemplate* erzeugt ein neues Objekt, das mit den Daten aus der angegebenen Datei initialisiert wird. Das erzeugte Dokument wird zum Bearbeiten geöffnet und in

die Client-Anwendung eingebettet, wenn es aktualisiert oder geschlossen wird.

Parameter

lpServer: Zeigt auf eine *OLESERVER*-Struktur, die den Server bezeichnet.

lhDoc: Bezeichnet das Dokument. Die Bibliothek verwendet dieses Handle intern.

lpszClass: Zeigt auf einen nullterminierten String, der die Klasse des zu erzeugenden Dokuments angibt.

lpszDoc: Zeigt auf einen nullterminierten String, der den Namen des Dokuments enthält, das erzeugt werden soll. Dieser Name kann vom Server in Fenstertiteln benutzt werden.

lpszTemplate: Zeigt auf einen nullterminierten String, der den Namen des Dokuments enthält, das zur Initialisierung des neuen Dokuments verwendet werden soll. Normalerweise ist dieser String ein Pfad, könnte aber für manche Anwendungen genauer angegeben sein. Der String könnte z.B. eine bestimmte Tabelle in einer Datenbank angeben.

lplpDoc: Zeigt auf eine Variable vom Typ *LPOLESERVERDOC*, in der die Server-Anwendung einen **long**-Zeiger auf die erzeugte *OLESERVERDOC*-Struktur zurückgeben sollte.

Rückgabewert

Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis

Wenn die Bibliothek diese Funktion aufruft, erzeugt die Server-Anwendung ein Dokument der angegebenen Klasse, reserviert und initialisiert eine *OLESERVERDOC*-Struktur, ordnet das Handle der Bibliothek diesem Dokument zu und liefert die Adresse der Struktur zurück.

Eine Server-Anwendung verfolgt meistens die im angegebenen Dokument gemachten Änderungen, so daß der Benutzer aufgefordert werden kann, die Änderungen zu speichern, falls dies notwendig ist.

Callback-Funktion

Edit

Deklaration

```
OLESTATUS (FAR PASCAL *Edit)(lpServer, lhDoc, lpszClass,
    lpszDoc, lplpDoc);

LPOLESERVER lpServer;        /* Handle für den Server */
LHSERVERDOC lhDoc;           /* Handle für das
                                Dokument              */
OLE_LPCSTR lpszClass;        /* Name der Klasse       */
OLE_LPCSTR lpszDoc;          /* Name des Dokuments    */
LPOLESERVERDOC FAR* lplpDoc; /* Platzhalter für die
                                zurückgegebene
                                OLESERVERDOC-Struktur */
```

Beschreibung Die Funktion *Edit* erzeugt ein Dokument, das mit Daten initialisiert wird, die durch einen vorhergehenden Aufruf von *SetData* abgerufen wurden. Das Objekt wird in die Client-Anwendung eingebettet. Der Server zeigt das Dokument und sein Fenster nicht an.

Parameter *lpServer:* Zeigt auf eine *OLESERVER*-Struktur, die den Server bezeichnet.

lhDoc: Bezeichnet das Dokument. Die Bibliothek verwendet dieses Handle intern.

lpszClass: Zeigt auf einen nullterminierten String, der die Klasse des zu erzeugenden Dokuments bestimmt.

lpszDoc: Zeigt auf einen nullterminierten String, der den Namen des Dokuments enthält, das erzeugt werden soll. Dieser Name muß nicht vom Server verwendet werden, kann aber z.B. in Fenstertiteln benutzt werden.

lplpDoc: Zeigt auf eine Variable vom Typ *LPOLESERVERDOC*, in der die Server-Anwendung einen **long-**Zeiger auf die erzeugte *OLESERVERDOC*-Struktur zurückgeben sollte.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis Wenn die Bibliothek diese Funktion aufruft, dann erzeugt die Server-Anwendung ein Dokument der angegebenen Klasse, reserviert und initialisiert eine *OLESERVERDOC*-Struktur, ordnet das Handle der Bibliothek diesem Dokument zu und liefert die Adresse der Struktur zurück.

Das durch die Funktion *Edit* erzeugte Dokument erhält seine Anfangsdaten vom Client durch einen nachfolgenden Aufruf der Funktion *SetData*. Der Benutzer kann das Dokument bearbeiten, nachdem die Daten abgerufen worden sind und die Bibliothek entweder die Funktion *Show* in der *OLEOBJECTVTBL*-Struktur oder die Funktion *DoVerb* mit der Aktionsart *Edit* verwendet hat, um das Dokument anzuzeigen.

Callback-Funktion **Execute**

Deklaration

```
OLESTATUS (FAR PASCAL *Execute)(lpServer, hCommands);

LPOLESERVER lpServer;        /* Handle für den Server */
HGLOBAL hCommands;           /* Puffer im Speicher mit
                                den Befehlen für den
                                Server               */
```

Beschreibung Die Funktion *Execute* erhält *WM_DDE_EXECUTE*-Befehle, die von Client-Anwendungen durch Aufrufe der Funktion *OleExecute* gesendet werden.

Parameter *lpServer:* Zeigt auf eine *OLESERVER*-Struktur, die den Server bezeichnet.

hCommands: Bezeichnet Speicher, der einen oder mehrere auszuführende DDE-Befehle enthält.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis Der Server sollte niemals das im Parameter *hCommands* angegebene Handle freigeben.

Callback-Funktion **Exit**

Deklaration

```
OLESTATUS (FAR PASCAL *Exit)(lpServer);

LPOLESERVER lpServer;   /* Handle für den Server      */
```

Beschreibung Die Funktion *Exit* weist die Server-Anwendung an, das Dokument zu schließen und zu beenden.

Parameter *lpServer:* Zeigt auf eine *OLESERVER*-Struktur, die den Server bezeichnet.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis Die Server-Bibliothek ruft die Funktion *Exit* auf, um eine Server-Anwendung, zu beenden. Wenn der Server beim Aufruf keine geöffneten Dokumente hat, sollte er die Funktion *OleRevokeServer* aufrufen.

Callback-Funktion **Open**

Deklaration

```
OLESTATUS (FAR PASCAL *Open)(lpServer, lhDoc, lpszDoc,
   lplpDoc);

LPOLESERVER lpServer;        /* Handle für den Server */
LHSERVERDOC lhDoc;           /* Handle für das
                                Dokument              */
OLE_LPCSTR lpszDoc;          /* Name des Dokuments    */
LPOLESERVERDOC FAR* lplpDoc; /* Platzhalter für die
                                zurückgegebene
                                OLESERVERDOC-Struktur */
```

Beschreibung Die Funktion *Open* öffnet eine vorhandene Datei und bereitet sie zum Bearbeiten vor. Sie wird normalerweise von einem Server zum Öffnen eines verknüpften Objekts für eine Client-Anwendung verwendet.

Parameter *lpServer:* Zeigt auf eine *OLESERVER*-Struktur, die den Server bezeichnet.

lhDoc: Bezeichnet das Dokument. Die Bibliothek verwendet dieses Handle intern.

lpszDoc: Zeigt auf einen nullterminierten String, der den permanenten Namen des Dokuments enthält, das geöffnet werden soll. Normalerweise ist dieser String ein Pfad, aber für manche Anwendungen könnte er genauer angegeben sein. Der String könnte z.B. eine bestimmte Tabelle in einer Datenbank angeben.

lplpDoc: Zeigt auf eine Variable vom Typ *LPOLESERVERDOC*, in der die Server-Anwendung einen **long**-Zeiger auf die *OLESERVERDOC*-Struktur zurückgibt, die sie als Antwort auf diese Funktion erzeugt hat.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis Wenn die Bibliothek diese Funktion aufruft, öffnet die Server-Anwendung das angegebene Dokument, reserviert und initialisiert eine *OLESERVERDOC*-Struktur, ordnet das Handle der Bibliothek diesem Dokument zu und liefert die Adresse der Struktur zurück. Der Server zeigt das Dokument und sein Fenster nicht an.

Callback-Funktion **Release**

Deklaration

```
OLESTATUS (FAR PASCAL *Release)(lpServer);

LPOLESERVER lpServer;        /* Handle für den Server */
```

Beschreibung Die Funktion *Release* benachrichtigt eine Server-Anwendung davon, daß alle Verbindungen zu ihr geschlossen sind und sie beendet werden kann.

Parameter *lpServer:* Zeigt auf eine *OLESERVER*-Struktur, die den Server bezeichnet.

Rückgabewert Der Rückgabewert ist *OLE_OK*, wenn die Funktion erfolgreich ist. Andernfalls wird ein Fehlerwert zurückgegeben.

Hinweis Die Server-Bibliothek ruft die Funktion *Release* auf, wenn ein Server beendet werden kann. Wenn eine Server-Anwendung die Funktion *OleRevokeServer* aufruft, muß die Anwendung weiterhin Botschaften senden und darauf warten, daß die Bibliothek die Funktion *Release* aufruft, bevor sie beendet.

Wenn der Server unsichtbar ist und die Bibliothek *Release* aufruft, muß er beenden. Die einzige Ausnahme ist, wenn die Anwendung mehrfach als Server dient. In diesem Fall wird der unsichtbare Server nicht notwendigerweise beendet, wenn die Bibliothek *Release* aufruft. Wenn der Server keine geöffneten Dokumente hat und mit der Option */Embedding* (die anzeigt, daß er von einer Client-Anwendung gestartet wurde) aufge-

rufen wurde, sollte er beenden, wenn die Bibliothek *Release* aufruft. Hat der Anwender jedoch ausdrücklich ein Dokument in einen Einfachinstanzen(MDI)-Server (Multidokumentschnittstelle) geladen, sollte der Server nicht beenden, wenn die Bibliothek *Release* aufruft. Normalerweise ist ein Einfachinstanzen-Server ein MDI-Server.

Alle registrierten Server-Strukturen müssen freigegeben werden, bevor ein Server beenden kann.

Ein Server kann die Funktion *PostQuitMessage* innerhalb der Funktion *Release* aufrufen.

OLESTREAM 3.1

Definition

```
#include <ole.h>

typedef struct _OLESTREAM { /* ostr */
    LPOLESTREAMVTBL lpstbl;
} OLESTREAM;
```

Beschreibung Die Struktur *OLESTREAM* zeigt auf eine *OLESTREAMVTBL*-Struktur, die Funktionen für die Stream-Eingabe und -Ausgabe bereitstellt. Diese Funktionen werden von der Client-Bibliothek für Stream-Operationen auf Objekte verwendet. Die *OLESTREAM*-Struktur wird von Client-Anwendungen reserviert und initialisiert.

Elemente *lpstbl:* Zeigt auf eine *OLESTREAMVTBL*-Struktur.

OLESTREAMVTBL 3.1

Definition

```
#include <ole.h>

typedef struct _OLESTREAMVTBL { /* ostrv */
    DWORD (CALLBACK* Get)(LPOLESTREAM, void FAR*,
      DWORD);
    DWORD (CALLBACK* Put)(LPOLESTREAM,
      OLE_CONST void FAR*, DWORD);
} OLESTREAMVTBL;
```

Beschreibung *OLESTREAMVTBL* zeigt auf Funktionen, die die Client-Bibliothek für Stream-Operationen auf Objekte verwendet. Sie wird von Client-Anwendungen reserviert und initialisiert.

Der Stream ist nur für die Dauer der Funktion gültig, an die er übergeben wird. Die Bibliothek fordert alle benötigten Informationen an, solange der Stream gültig ist.

Die Rückgabewerte der Stream-Funktionen können anzeigen, daß ein Fehler aufgetreten ist, aber nicht, warum er aufgetreten ist. Die Client-Anwendung ist für alle notwendigen fehlerermittelnden Operationen verantwortlich.

Eine Client-Anwendung kann diese Funktionen verwen-

den, um Variationen der Standard-Stream-Funktionen bereitzustellen. Der Client könnte z.B. das permanente Speichern einiger Objekte so ändern, daß sie in einer Datenbank anstelle des Client-Dokuments gespeichert würden.

Callback-Funktion **Get**

Deklaration

```
DWORD Get(lpstream, lpszBuf, cbbuf);

LPOLESTREAM lpstream;       /* Eingabe-Stream       */
void FAR* lpszBuf;          /* Puffer für die Daten */
DWORD cbbuf;                /* Anzahl der zu lesenden
                               Byte                 */
```

Beschreibung Die Funktion *Get* liest Daten aus dem angegebenen Stream.

Parameter *lpstream:* Zeigt auf eine vom Client reservierte *OLESTREAM*-Struktur.

lpszBuf: Zeigt auf einen Puffer, der mit Daten aus dem Stream gefüllt werden soll.

cbbuf: Legt die Anzahl der Byte fest, die in den Puffer eingelesen werden.

Rückgabewert Der Rückgabewert ist die Anzahl der tatsächlich in den Puffer eingelesenen Byte, wenn die Funktion erfolgreich ist. Wenn das Dateiende erreicht ist, wird Null zurückgegeben. Ein negativer Wert weist auf einen Fehler hin.

Hinweis Der vom Parameter *cbbuf* angegebene Wert kann größer als 64 KByte sein. Wenn die Client-Anwendung eine stream-lesende Funktion verwendet, die auf 64 KByte begrenzt ist, sollte sie die Funktion wiederholt aufrufen, bis sie die von *cbbuf* angegebene Anzahl eingelesen hat. Immer wenn die Daten größer als 64 KByte sind, befindet sich der Zeiger auf den Datenpuffer am Anfang des Segments.

Callback-Funktion **Put**

Deklaration

```
DWORD Put(lpstream, lpszBuf, cbbuf);

LPOLESTREAM lpstream;       /* Eingabe-Stream       */
OLE_CONST void FAR* lpszBuf; /* Puffer für die Daten */
DWORD cbbuf;                /* Anzahl der zu lesenden
                               Byte                 */
```

Beschreibung Die Funktion *Put* schreibt Daten in den angegebenen Stream.

Parameter *lpstream:* Zeigt auf eine vom Client reservierte *OLESTREAM*-Struktur.

lpszBuf: Zeigt auf einen Puffer, aus dem die Daten in den Stream geschrieben werden sollen.

cbbuf: Legt die Anzahl der Byte fest, die in den Stream geschrieben werden sollen.

Rückgabewert Der Rückgabewert ist die Anzahl der tatsächlich in den Stream geschriebenen Byte. Ein Rückgabewert, der kleiner als der in *cbbuf* angegebene Wert ist, zeigt an, daß nicht genügend Platz im Stream war oder ein Fehler aufgetreten ist.

Hinweis Der vom Parameter *cbbuf* angegebene Wert kann größer als 64 KByte sein. Wenn die Client-Anwendung eine stream-schreibende Funktion verwendet, die auf 64 KByte begrenzt ist, sollte sie die Funktion wiederholt aufrufen, bis sie die von *cbbuf* angegebene Anzahl Byte geschrieben hat. Überschreitet die Größe der Daten 64 KByte, befindet sich der Zeiger auf den Datenpuffer am Anfang des Segments.

OLETARGETDEVICE 3.1

Definition

```
#include <ole.h>

typedef struct _OLETARGETDEVICE {
    UINT otdDeviceNameOffset;
    UINT otdDriverNameOffset;
    UINT otdPortNameOffset;
    UINT otdExtDevmodeOffset;
    UINT otdExtDevmodeSize;
    UINT otdEnvironmentOffset;
    UINT otdEnvironmentSize;
    BYTE odtData[1];
} OLETARGETDEVICE;
```

Beschreibung Die Struktur *OLETARGETDEVICE* enthält Informationen über das Zielgerät, das eine Client-Anwendung benutzt. Server-Anwendungen können durch die Informationen in dieser Struktur die Übergabe eines Objekts ändern, falls dies nötig ist. Eine Client-Anwendung stellt das Handle dieser Struktur in einem Aufruf der Funktion *OleSetTargetDevice* bereit.

Elemente *otdDeviceNameOffset:* Gibt den Offset vom Anfang des Array zum Namen des Geräts an.

otdDriverNameOffset: Gibt den Offset vom Anfang des Array zum Namen des Gerätetreibers an.

otdPortNameOffset: Gibt den Offset vom Anfang des Array zum Namen der Schnittstelle an.

otdExtDevmodeOffset: Gibt den Offset vom Anfang des Array zu einer *DEVMODE*-Struktur an, die mit der Funktion *ExtDeviceMode* abgerufen wurde.

otdExtDevmodeSize: Gibt die Größe der *DEVMODE*-Struktur an, deren Offset durch das Element *otdExtDevmodeOffset* festgelegt wird.

otdEnvironmentOffset: Bestimmt den Offset vom Anfang des Array zur Umgebung der Geräteschnittstelle.

otdEnvironmentSize: Gibt die Größe der Umgebung an, deren Offset durch das Element *otdEnvironmentOffset* festgelegt wird.

odtData: Gibt ein Byte-Array an, das Daten für das Zielgerät enthält.

Hinweis

Die Elemente *otdDeviceNameOffset, otdDriverNameOffset* und *otdPortNameOffset* sollten nullterminiert sein.

In Windows 3.1 ist es durch die Fähigkeit, mehrere Drucker an eine Schnittstelle anzuschließen, nicht mehr korrekt, die Umgebungsinformation zu verwenden. Die von der Funktion *GetEnvironment* gelieferten Umgebungsinformationen können gelegentlich falsch sein. Um sicherzustellen, daß die *OLETARGETDEVICE*-Struktur richtig initialisiert wird, sollte die Anwendung die Informationen aus der *DEVMODE*-Struktur, die durch einen Aufruf der Funktion *ExtDeviceMode* abgerufen wurde, an die Position der Umgebungsinformation der *OLETARGETDEVICE*-Struktur kopieren.

Siehe auch

OleSetTargetDevice

OPENFILENAME 3.1

Definition

```
#include <commdlg.h>

typedef struct tagOPENFILENAME {                /* ofn */
   DWORD lStructSize;
   HWND hwndOwner;
   HINSTANCE hInstance;
   LPCSTR lpstrFilter;
   LPSTR lpstrCustomFilter;
   DWORD nMaxCustFilter;
   DWORD nFilterIndex;
   LPSTR lpstrFile;
   DWORD nMaxFile;
   LPSTR lpstrFileTitle;
   DWORD nMaxFileTitle;
   LPCSTR lpstrInitialDir;
   LPCSTR lpstrTitle;
   DWORD Flags;
   UINT nFileOffset;
   UINT nFileExtension;
   LPCSTR lpstrDefExt;
   LPARAM lCustData;
```

```
    UINT (CALLBACK *lpfnHook)(HWND, UINT, LPARAM,
          WPARAM);
    LPCSTR lpTemplateName;
} OPENFILENAME;
```

Beschreibung Die Struktur *OPENFILENAME* enthält Informationen, die das System verwendet, um das systemdefinierte Dialogfenster *Datei öffnen* oder *Datei speichern* zu initialisieren. Nachdem der Benutzer den Schalter *OK* gewählt hat, um das Dialogfenster zu schließen, gibt das System Informationen über die Auswahl des Benutzers in dieser Struktur zurück.

Elemente *lStructSize:* Gibt die Größe der Struktur in Byte an. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

hwndOwner: Bezeichnet das Fenster, zu dem das Dialogfenster gehört. Dieses Element kann jedes gültige Fenster-Handle oder *NULL* sein, wenn das Dialogfenster zu keinem Fenster gehört.

Wenn das Flag *OFN_SHOWHELP* gesetzt ist, muß *hwndOwner* das Fenster bezeichnen, zu dem das Dialogfenster gehört. Die Fensterfunktion dieses besitzenden Fensters erhält eine Benachrichtigungsbotschaft, wenn der Benutzer den *Hilfe*-Schalter wählt. Der Bezeichner der Benachrichtigungsbotschaft ist der Wert, der von der Funktion *RegisterWindowMessage* zurückgegeben wird, wenn ihr *HELPMSGSTRING* als Parameter übergeben wird. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

hInstance: Bezeichnet einen Datenblock, der eine Dialogfensterschablone enthält, die vom Element *lpTemplateName* festgelegt wird. Dieses Element wird nur verwendet, wenn das Element *Flags* das *OFN_ENABLETEMPLATE-* oder *OFN_ENABLETEMPLATEHANDLE*-Flag enthält; andernfalls wird dieses Element ignoriert. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

lpstrFilter: Zeigt auf einen Puffer, der ein oder mehrere Paare nullterminierter Strings enthält, die Filter festlegen. Der erste String in jedem Paar beschreibt einen Filter (z.B. "Textdateien"), der zweite bestimmt das Filtermuster (z.B. "*.txt"). Mehrere Filter können für ein Element angegeben werden; in diesem Fall wird das Semikolon (;) verwendet, um Filtermuster-Strings zu trennen, z.B. "*.txt;*.doc;*.bak". Der letzte String im Puffer muß mit zwei Nullzeichen abgeschlossen werden. Wenn dieses Element *NULL* ist, zeigt das Dialogfenster keine Filter an. Die Filter-Strings müssen in der richtigen Reihenfolge vorliegen – das System ändert die Reihenfolge nicht. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

lpstrCustomFilter: Zeigt auf einen Puffer, der ein Paar benutzerdefinierter Strings enthält, die einen Filter festlegen. Der erste String beschreibt den Filter und der zweite das Filtermuster (z.B. "WinWord", "*.doc"). Der Puffer wird mit zwei Nullzeichen abgeschlossen. Das System kopiert die Strings in den Puffer, wenn der Anwender den *OK*-Schalter wählt, um das Dialogfenster zu schließen. Das System verwendet die Strings als anfängliche Filterbeschreibung und -muster für das Dialogfenster. Wenn dieser Parameter *NULL* ist, listet das Dialogfenster benutzerdefinierte Filter-Strings auf, speichert sie aber nicht.

nMaxCustFilter: Gibt die Größe des Puffers in Byte an, den das Element *lpstrCustomFilter* bezeichnet. Der Puffer sollte mindestens 40 Byte lang sein. Dieser Parameter wird ignoriert, wenn das Element *lpstrCustomFilter NULL* ist. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

nFilterIndex: Gibt einen Index in den Puffer an, auf den das Element *lpstrFilter* zeigt. Das System kann durch diesen Indexwert ein String-Paar ermitteln, das als anfängliche Filterbeschreibung und -muster für das Dialogfenster verwendet wird. Das erste Paar hat den Index 1. Wenn der Anwender *OK* wählt, um das Dialogfenster zu schließen, kopiert das System den Index des ausgewählten Filter-String an diese Position.

Wenn das Element *nFilterIndex* den Wert 0 hat, wird der Filter in dem Puffer verwendet, auf den das Element *lpstrCustomFilter* zeigt. Wenn *nFilterIndex* 0 und das Element *lpstrCustomFilter NULL* ist, verwendet das System den ersten Filter im Puffer, auf den das Element *lpstrFilter* zeigt. Wenn jedes der drei Elemente entweder 0 oder *NULL* ist, verwendet das System überhaupt keine Filter und zeigt keine Dateien in der Liste *Dateiname* im Dialogfenster an.

lpstrFile: Zeigt auf einen Puffer, der einen Dateinamen enthält. Dieser wird verwendet, um das Editierfeld *Dateiname* zu initialisieren. Wenn keine Initialisierung erforderlich ist, muß das erste Zeichen dieses Puffers *NULL* sein. Wenn die *GetOpenFileName*- oder *GetSaveFileName*-Funktion zurückkehrt, enthält dieser Puffer den vollständigen Ort und Namen der ausgewählten Datei.

Wenn der Puffer zu klein ist, kopiert die Dialogfensterfunktion die erforderliche Größe in dieses Element und gibt 0 zurück. Um die benötigte Größe zu ermitteln, muß der Typ des Elements *lpstrFile* in *LPWORD* umgewandelt werden. Der Puffer muß mindestens drei Byte lang sein, um den Größenwert aufzunehmen. Wenn er zu klein ist, liefert die Funktion *CommDlgExtendedError* den Wert *FNERR_BUFFERTOOSMALL* zurück.

nMaxFile: Gibt die Größe des Puffers in Byte an, auf den das Element *lpstrFile* zeigt. Die Funktionen *GetOpenFileName* und *GetSaveFileName* geben *FALSE* zurück, wenn der Puffer zu klein ist, um die Dateiinformationen aufzunehmen. Der Puffer sollte mindestens 256 Byte lang sein. Dieses Element wird ignoriert, wenn das Element *lpstrFile NULL* ist. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

lpstrFileTitle: Zeigt auf einen Puffer, der den Titel der ausgewählten Datei aufnimmt. Dieser Puffer nimmt den Dateinamen und die Namenserweiterung auf, aber keine Pfadinformationen. Eine Anwendung sollte diesen String verwenden, um den Dateinamen anzuzeigen. Wenn dieses Element *NULL* ist, kopiert die Funktion den Dateinamen nicht. Dieses Element wird beim Schließen des Fensters gesetzt.

nMaxFileTitle: Gibt die maximale Länge des String in Byte an, der in den *lpstrFileTitle*-Puffer kopiert werden kann. Dieses Element wird ignoriert, wenn *lpstrFileTitle NULL* ist. Es wird für die Initialisierung des Dialogfensters benötigt.

lpstrInitialDir: Zeigt auf einen String, der das Anfangsverzeichnis bestimmt. Wenn dieses Element *NULL* ist, verwendet das System das aktuelle Verzeichnis. Wenn das Element *lpstrFile* einen String enthält, der einen gültigen Pfad angibt, verwendet die Standarddialogfensterfunktion diesen Pfad anstelle des Pfads in dem String, auf den *lpstrInitialDir* zeigt.

Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

lpstrTitle: Zeigt auf einen String, der in der Titelleiste des Dialogfensters erscheinen soll. Wenn dieses Element *NULL* ist, verwendet das System den voreingestellten Titel, d.h. *Datei speichern unter* oder *Datei öffnen*. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

Flags: Legt die Initialisierungs-Flags des Dialogfensters fest. Dieses Element kann eine Kombination der folgenden Werte sein:

Wert	Bedeutung
OFN_ALLOWMULTI-SELECT	Legt fest, daß die Liste *Dateiname* die Auswahl mehrerer Dateien zulassen soll. Wenn dieses Flag gesetzt ist, zeigt das Element *lpstrFile* auf einen Puffer, der den Pfad des aktuellen Verzeichnisses und alle ausgewählten Dateinamen enthält. Der erste Dateiname ist durch ein Leerzeichen vom Pfad getrennt.

Wert	Bedeutung

	Jeder nachfolgende Dateiname ist durch ein Leerzeichen vom vorhergehenden getrennt. Einigen ausgewählten Dateinamen kann ein relativer Pfad voranstehen. Der Puffer könnte z.B. folgendes enthalten:
	`c:\files file1.txt file2.txt`
	`..\bin\file3.txt`

OFN_CREATPROMPT	Veranlaßt die Dialogfensterfunktion, ein Meldungsfenster zu erzeugen, um den Benutzer zu benachrichtigen, wenn eine bestimmte Datei nicht vorhanden ist. So kann der Benutzer entscheiden, ob die Datei erzeugt werden soll. (Dieses Flag setzt automatisch die Flags *OFN_PATHMUSTEXIST* und *OFN_FILEMUSTEXIST*.)
OFN_ENABLEHOOK	Aktiviert die im Element *lpfnHook* angegebene Filterfunktion.
OFN_ENABLE-TEMPLATE	Veranlaßt das System, die durch die Elemente *hInstance* und *lpTemplateName* bezeichnete Dialogfensterschablone zur Erzeugung des Dialogfensters zu verwenden.
OFN_ENABLE-TEMPLATE-HANDLE	Zeigt an, daß das Element *hInstance* einen Datenblock bezeichnet, der eine geladene Dialogfensterschablone enthält. Das System ignoriert das Element *lpTemplateName*, wenn dieses Flag angegeben wird.
OFN_EXTENSION-DIFFERENT	Zeigt an, daß die Dateinamenserweiterung des zurückgelieferten Dateinamens verschieden von der im Element *lpstr-DefExt* angegebenen ist. Dieses Flag wird nicht gesetzt, wenn *lpstrDefExt NULL* ist, wenn die Namenserweiterungen übereinstimmen oder wenn die Datei keine Namenserweiterung hat. Dieses Flag kann beim Schließen des Fensters gesetzt werden.
OFN_FILEMUSTEXIST	Legt fest, daß der Anwender nur Namen von vorhandenen Dateien im Editierfeld *Dateiname* eingeben kann.

Wert	Bedeutung
	Ist dieses Flag gesetzt und der Anwender gibt einen ungültigen Dateinamen ein, gibt die Dialogfensterfunktion eine Warnung in einem Meldungsfenster aus. Dieses Flag bewirkt außerdem, daß das *OFN_PATHMUSTEXIST*-Flag gesetzt wird.
OFN_HIDEREADONLY	Verbirgt das Auswahlfeld *Schreibgeschützt*.
OFN_NOCHANGEDIR	Veranlaßt das Dialogfenster, das aktuelle Verzeichnis auf das bei seiner Erzeugung festgelegte zurückzusetzen.
OFN_NOREADONLY-RETURN	Legt fest, daß bei der zurückgelieferten Datei das Attribut Nur-Lesen nicht gesetzt ist und daß sie sich nicht in einem schreibgeschützten Verzeichnis befindet.
OFN_NOTESTFILE-CREATE	Legt fest, daß die Datei nicht erzeugt wird, bevor das Dialogfenster geschlossen ist. Dieses Flag sollte gesetzt werden, wenn die Anwendung die Datei auf einem Netzknoten mit dem Attribut Erzeugen/Nicht-Ändern speichert. Wenn eine Anwendung dieses Flag setzt, überprüft die Bibliothek nicht auf Schreibschutz, ein volles Laufwerk, eine geöffnete Laufwerkstür oder Netzschutz. Deshalb müssen Anwendungen, die dieses Flag verwenden, Dateioperationen vorsichtig verwenden – eine Datei kann nicht erneut geöffnet werden, wenn sie einmal geschlossen ist.
OFN_NOVALIDATE	Gibt an, daß die Standarddialogfenster ungültige Zeichen in den zurückgegebenen Dateinamen zulassen. Normalerweise verwendet die aufrufende Anwendung eine Filterfunktion, die die Dateinamen unter Verwendung der registrierten Botschaft *FILEOKSTRING* überprüft. Wenn kein Text oder nur Leerzeichen im Editierfeld stehen, werden die Listen der Dateien und Verzeichnisse aktualisiert. Wenn der Text im Editierfeld etwas anderes enthält, werden die Elemente *nFileOffset* und *nFileExtension* auf Werte gesetzt, die durch Analysieren des Texts erzeugt wurden.

Wert	Bedeutung
	Es wird weder die voreingestellte Dateinamenserweiterung zum Text hinzugefügt noch der Text in den *lpstrFileTitle*-Puffer kopiert. Wenn der vom Element *nFileOffset* angegebene Wert negativ ist, ist der Dateiname ungültig. Wenn er nicht negativ ist, ist der Dateiname gültig und *nFileOffset* und *nFileExtension* können verwendet werden, als ob das *OFN_NOVALIDATE*-Flag nicht gesetzt worden wäre.
OFN_OVERWRITE-PROMPT	Veranlaßt das Dialogfenster *Datei speichern unter* ein Meldungsfenster zu erzeugen, wenn die ausgewählte Datei bereits vorhanden ist. Der Benutzer muß bestätigen, ob die Datei überschrieben werden soll.
OFN_PATHMUST-EXIST	Legt fest, daß der Anwender nur gültige Pfadnamen im Editierfeld *Dateiname* eingeben kann. Wenn dieses Flag gesetzt ist und der Anwender einen ungültigen Pfadnamen eingibt, gibt die Dialogfensterfunktion eine Warnung in einem Meldungsfenster aus.
OFN_READONLY	Das Auswahlfeld *Schreibgeschützt* ist bei der Erzeugung des Dialogfensters markiert. Wenn der Benutzer den *OK*-Schalter wählt, um das Dialogfenster zu schließen, wird der Status des Auswahlfelds von diesem Element bestimmt. Dieses Flag wird für die Initialisierung des Dialogfensters benötigt und kann beim Schließen des Fensters gesetzt werden.
OFN_SHAREAWARE	Wenn ein Aufruf der Funktion *OpenFile* wegen einer Zugriffsverletzung bei der gemeinsamen Nutzung von Dateien fehlgeschlagen ist, wird der Fehler ignoriert und das Dialogfenster gibt den angegebenen Dateinamen zurück. Wenn dieses Flag nicht gesetzt ist, wird die registrierte Botschaft SHAREVISTRING an die Filterfunktion gesendet und dabei ein Zeiger auf einen nullterminierten String mit dem Pfadnamen im Parameter *lParam* übergeben.

Wert	Bedeutung
	Die Filterfunktion anwortet mit einem der folgenden Werte: *OFN_SHAREFALLTHROUGH:* Gibt an, daß der Dateiname vom Dialogfenster abgewiesen wurde. *OFN_SHARENOWARN:* Gibt keine weitere Aktion an. *OFN_SHAREWARN:* Legt fest, daß der Benutzer die Standardwarnung für diesen Fehler erhält. (Dies führt zum gleichen Ergebnis, als wäre keine Filterfunktion vorhanden.) Dieses Flag kann beim Schließen des Fensters gesetzt werden.
OFN_SHOWHELP	Veranlaßt das Dialogfenster, den *Hilfe*-Schalter anzuzeigen. Das Element *hwndOwner* darf nicht *NULL* sein, wenn diese Option angegeben wird.

Diese Flags können – wenn nicht anders angegeben – gesetzt werden, wenn die Struktur initialisiert wird.

nFileOffset: Bestimmt einen von Null ausgehenden Offset vom Anfang des Pfads zum Dateinamen, der von dem String im Puffer angegeben wird, auf den *lpstrFile* zeigt. Wenn z.B. *lpstrFile* auf den String "c:\dir1\dir2\file.ext" zeigt, enthält dieses Element den Wert 13. Dieses Element wird beim Schließen des Fensters gesetzt.

nFileExtension: Bestimmt einen von Null ausgehenden Offset vom Anfang des Pfads zur Dateinamenserweiterung, die von dem String im Puffer angegeben wird, auf den *lpstrFile* zeigt. Wenn z.B. *lpstrFile* auf den String "c:\dir1\dir2\file.ext" zeigt, enthält dieses Element den Wert 18. Wenn der Benutzer keine Namenserweiterung eingegeben hat und *lpstrDefExt NULL* ist, legt dieses Element einen Offset zum abschließenden Nullzeichen fest. Wenn ein Punkt (.) als letztes Zeichen im Dateinamen eingegeben wird, ist dieses Element 0. Dieses Element wird beim Schließen des Fensters gesetzt.

lpstrDefExt: Zeigt auf einen Puffer, der die voreingestellte Dateinamenserweiterung enthält. Die Funktionen *GetOpenFileName* und *GetSaveFileName* hängen diese Namenserweiterung an den Dateinamen an, wenn der Benutzer keine angegeben hat. Dieser String kann beliebig lang sein, es werden aber nur die ersten drei Zeichen angehängt. Der String sollte keinen Punkt (.) enthalten. Wenn dieses Element *NULL* ist und der Benutzer keine Namenserweiterung eingibt, wird keine Na-

menserweiterung angehängt. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

lCustData: Bestimmt anwendungsdefinierte Daten, die das System an die Filterfunktion übergibt, auf die das Element *lpfnHook* zeigt. Das System übergibt einen Zeiger auf die *OPENFILENAME*-Struktur im Parameter *lParam* der *WM_INITDIALOG*-Botschaft. Durch diesen Parameter kann das Element *lCustData* abgerufen werden.

lpfnHook: Zeigt auf eine Filterfunktion, die Botschaften für das Dialogfenster bearbeitet. Um die Filterfunktion zu aktivieren, muß eine Anwendung das Flag *OFN_ENABLEHOOK* im Element *Flags* angeben; andernfalls ignoriert das System dieses Strukturelement. Die Filterfunktion muß Null zurückgeben, um eine Botschaft, die sie nicht bearbeitet hat, an die Dialogfensterfunktion in COMMDLG.DLL zu übergeben. Die Filterfunktion muß einen Wert ungleich Null zurückliefern, um die Dialogfensterfunktion in COMMDLG.DLL davon abzuhalten, eine bereits von ihr bearbeitete Botschaft zu bearbeiten.

Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

lpTemplateName: Zeigt auf einen nullterminierten String, der den Namen der Ressourcendatei für die Dialogfensterschablone angibt, die anstelle der Dialogfensterschablone in COMMDLG.DLL verwendet werden soll. Eine Anwendung kann das Makro MAKEINTRESOURCE für numerierte Dialogfensterressourcen verwenden. Dieses Element wird nur verwendet, wenn das Element *Flags* das Flag *OFN_ENABLETEMPLATE* enthält, andernfalls wird es ignoriert.

Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

Siehe auch *GetOpenFileName, GetSaveFileName*

OUTLINETEXTMETRIC 3.1

Definition

```
typedef struct tagOUTLINETEXTMETRIC {
  UINT otmSize;                /* Größe dieser
                                  Struktur          */
  TEXTMETRIC otmTextMetrics;   /* TEXTMETRIC-
                                  Struktur          */
  BYTE otmFiller;              /* Ausrichtung der
                                  Struktur an
                                  ungeraden Adressen */
  PANOSE otmPanoseNumber;      /* Panose-Nummer der
                                  Schrift           */
  UINT otmfsSelection;         /* Flags für die
                                  Schriftart        */
```

```
UINT otmfsType;             /* Typindikatoren      */
UINT otmsCharSlopeRise;     /* Neigungswinkel      */
UINT otmsCharSlopeRun;      /* Neigungswinkel      */
UINT otmItalicAngle;        /* Kursivneigung       */
UINT otmEMSquare;           /* Größe von EM        */
INT otmAscent;              /* Oberlänge über
                               Grundlinie          */
INT otmDescent;             /* Unterlänge unter
                               Grundlinie          */
UINT otmLineGap;            /* typografischer
                               Zeilenabstand       */
UINT otmsXHeight;           /* nicht unterstützt   */
UINT otmsCapEmHeight;       /* nicht unterstützt   */
RECT otmrcFontBox;          /* umgebender Felder
                               der Schrift         */
INT otmMacAscent;           /* Oberlänge über
                               Grundlinie für
                               MacIntosh           */
INT otmMacDescent;          /* Unterlänge unter
                               Grundlinie für
                               MacIntosh           */
UINT otmMacLineGap;         /* Zeilenabstand
                               für MacIntosh       */
UINT otmusMinimumPPEM;      /* Kleinste Schrift-
                               größe               */
POIN otmptSubscriptSize;    /* Größe tief-
                               gestellter Schrift  */
POIN otmptSubscriptOffset;  /* Offset tief-
                               gestellter Schrift  */
POIN otmptSuperscriptSize;  /* Größe hoch-
                               gestellter Schrift  */
POIN otmptSuperscriptOffset; /* Offset hoch-
                               gestellter Schrift  */
UINT otmsStrikeoutSize;     /* Durchstreichen-
                               Größe               */
INT otmsStrikeoutPosition;  /* Durchstreichen-
                               Position            */
INT otmsUnderscorePosition; /* Unterstrich-
                               Position            */
UINT otmsUnderscoreSize;    /* Unterstrich-Größe   */
PSTR otmpFamilyName;        /* Offset für den
                               Familiennamen
                               der Schrift         */
PSTR otmpFaceName;          /* Offset für den
                               Schriftnamen        */
PSTR otmpStyleName;         /* Offset für den
                               Schriftartnamen     */
```

```
  PSTR otmpFullName;          /* Offset für den
                                 vollständigen Namen
                                 der Schrift        */
} OUTLINETEXTMETRIC;
```

Beschreibung Die *OUTLINETEXTMETRIC*-Struktur enthält Maße, die eine TrueType-Schrift beschreiben.

Elemente *otmSize:* Gibt die Größe der *OUTLINETEXTMETRIC*-Struktur in Byte an.

otmTextMetrics: Gibt eine *TEXTMETRIC*-Struktur an, die weitere Informationen über die Schrift enthält.

otmFiller: Bestimmt einen Wert, der die Ausrichtung der Struktur an ungeraden Adressen veranlaßt.

otmPanoseNumber: Legt die Panose-Nummer dieser Schrift fest.

otmfsSelection: Gibt die Art des Schriftmusters an. Dieses Element kann eine Kombination der folgenden Bits sein:

Bit	Bedeutung
0	Kursiv
1	Unterstrichen
2	Negativ
3	Outline
4	Durchgestrichen
5	Fett

otmfsType: Gibt an, ob die Schrift lizensiert ist. Lizensierte Schriften können nicht geändert oder ausgetauscht werden. Wenn Bit 1 nicht gesetzt ist, kann die Schrift eingebettet werden. Wenn Bit 2 gesetzt ist, wird die Schrift mit dem Attribut Nur-Lesen eingerichtet.

otmsCharSlopeRise : Bestimmt die Neigung des Cursor. Dieser Wert ist 1, wenn die Neigung vertikal ist. Anwendungen können diesen Wert und den des Elements *otmsCharSlopeRun* verwenden, um einen kursiven Cursor zu erzeugen, der die gleiche Neigung wie der Hauptkursivwinkel (festgelegt durch das Element *otmItalicAngle*) hat.

otmsCharSlopeRun: Bestimmt die Neigung des Cursors. Dieser Wert ist Null, wenn die Neigung vertikal ist. Anwendungen können diesen Wert und den des Elements *otmsCharSlopeRise* verwenden, um einen kursiven Cursor zu erzeugen, der die gleiche Neigung wie der Hauptkursivwinkel (festgelegt durch das Element *otmItalicAngle*) hat.

otmItalicAngle: Legt den Hauptkursivwinkel der Schriftart fest (in Grad gegen den Uhrzeigersinn von der Vertikalen aus-

gehend). Regelmäßige (Roman) Schriften haben einen Wert von Null. Kursive Schriften haben normalerweise einen negativen Kursivwinkel (sie neigen sich nach rechts).

otmEMSquare: Gibt die Anzahl der virtuellen Einheiten an, die die x- oder y-Abmessungen des em-Quadrats dieser Schrift definieren. Die Anzahl der Einheiten in die x- und y-Richtung sind bei einem em-Quadrat immer gleich.

otmAscent: Legt den maximalen Abstand fest, um den Zeichen in dieser Schrift über die Grundlinie hinausragen. Dies ist die typografische Oberlänge dieser Schrift.

otmDescent: Legt den maximalen Abstand fest, um den Zeichen in dieser Schrift unter die Grundlinie hinausragen. Dies ist die typografische Unterlänge dieser Schrift.

otmLineGap: Bestimmt den typografischen Zeilenabstand.

otmXHeight: Nicht unterstützt.

otmCapEmHeight: Nicht unterstützt.

otmrcFontBox: Gibt das umgebende Feld für die Schrift an.

otmMacAscent: Legt den maximalen Abstand für den MacIntosh fest, um den Zeichen in dieser Schrift über die Grundlinie hinausragen.

otmMacDescent: Legt den maximalen Abstand für den MacIntosh fest, um den Zeichen in dieser Schrift unter die Grundlinie hinausragen.

otmMacLineGap: Gibt Zeilenabstandinformationen für den MacIntosh an.

otmusMinimumPPEM: Gibt die kleinste empfohlene Größe in Pixel pro em-Quadrat für diese Schrift an.

otmptSubscriptSize: Gibt die empfohlene horizontale und vertikale Größe für tiefgestellte Zeichen in dieser Schrift an.

otmptSubscriptOffset: Gibt den empfohlenen horizontalen und vertikalen Offset für tiefgestellte Zeichen in dieser Schrift an. Dieser Offset wird vom Zeichenursprung zum Ursprung des tiefgestellten Zeichens gemessen.

otmptSuperscriptSize: Gibt die empfohlene horizontale und vertikale Größe für hochgestellte Zeichen in dieser Schrift an.

otmptSuperscriptOffset: Gibt den empfohlenen horizontalen und vertikalen Offset für hochgestellte Zeichen in dieser Schrift an. Dieser Offset wird von der Zeichengrundlinie zur Grundlinie des hochgestellten Zeichens gemessen.

otmsStrikeoutSize: Bestimmt die Breite der Durchstreichen-Linie dieser Schrift. Normalerweise ist dies die Breite des em-Gedankenstrichs dieser Schrift.

otmsStrikeoutPosition: Gibt die Position der Durchstreichen-Linie relativ zur Grundlinie für diese Schrift an. Positive Werte sind oberhalb und negative unterhalb der Grundlinie.

otmsUnderscorePosition: Gibt die Position des Unterstrichzeichens für diese Schrift an.

otmsUnderscoreSize: Gibt die Dicke des Unterstrichzeichens für diese Schrift an.

otmpFamilyName: Bestimmt den Offset vom Anfang der Struktur zu einem String, der den Familiennamen der Schrift angibt.

otmpFaceName: Bestimmt den Offset vom Anfang der Struktur zu einem String, der den Schriftnamen angibt. Dieser Name entspricht dem in der *LOGFONT*-Struktur angegebenen Namen.

otmpStyleName: Bestimmt den Offset vom Anfang der Struktur zu einem String, der den Schriftartnamen angibt.

otmpFullName: Bestimmt den Offset vom Anfang der Struktur zu einem String, der den vollständigen Namen der Schrift angibt. Dieser Name ist eindeutig und enhält oft eine Versionsnummer oder andere beschreibende Informationen.

Hinweis

Die in der *OUTLINETEXTMETRIC*-Struktur zurückgelieferten Maße sind in virtuellen Einheiten angegeben, d.h. sie sind vom aktuellen Abbildungsmodus des angegebenen Gerätekontexts abhängig.

Siehe auch

GetOutlineTextMetrics

PANOSE 3.1

Definition

```
typedef struct tagPANOSE { /* panose */
   BYTE bFamilyType;
   BYTE bSerifStyle;
   BYTE bWeight;
   BYTE bProportion;
   BYTE bContrast;
   BYTE bStrokeVariation;
   BYTE bArmStyle;
   BYTE bLetterform;
   BYTE bMidline;
   BYTE bXHeight;
} PANOSE;
```

Beschreibung Die *PANOSE*-Struktur beschreibt die Panose-Schriftklassifizierungswerte für eine TrueType-Schrift.

Elemente *bFamilyType:* Bestimmt die Schriftfamilie. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
0	Beliebig
1	Nicht geeignet
2	Text und Anzeige
3	Handschrift
4	Dekorativ
5	Illustriert

bSerifStyle: Bestimmt den Serifenstil für diese Schrift. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
0	Beliebig
1	Nicht geeignet
2	Wölbung
3	Stumpfe Wölbung
4	Quadratische Wölbung
5	Stumpfe quadratische Wölbung
6	Quadrat
7	Dünn
8	Gräte
9	Übertrieben
10	Dreieck
11	Normal, ohne Serifen
12	Stumpf, ohne Serifen
13	$$Perp sans
14	$$Flared
15	Gerundet

bWeight: Bestimmt die Strichstärke der Schrift. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
0	Beliebig
1	Nicht geeignet
2	Sehr dünn
3	Dünn
4	Dünn
5	$$Book
6	Mittel
7	Halb

Wert	Bedeutung
8	Dick
9	Sehr dick
10	Schwarz
11	$$Nord

bProportion: Gibt die Proportion der Schrift an. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
0	Beliebig
1	Nicht geeignet
2	Old Style
3	Modern
4	Gleichmäßige Breite
5	Erweitert
6	Gedrängt
7	Sehr erweitert
8	Sehr gedrängt
9	Feste Zeichengrößen

bContrast: Legt den Kontrast der Schrift fest. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
0	Beliebig
1	Nicht geeignet
2	Keiner
3	Sehr niedrig
4	Niedrig
5	Mittel-niedrig
6	Mittel
7	Mittel-hoch
8	Hoch
9	Sehr hoch

bStrokeVariation: Bestimmt die Strichvariation dieser Schrift. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
0	Irgendeine
1	Nicht geeignet
2	Stufen/diagonal
3	Stufen/Übergang
4	Stufen/vertikal
5	Stufen/horizontal

Wert	Bedeutung
6	$$Rapid/vertikal
7	$$Rapid/horizontal
8	$$Instant/vertikal

bArmStyle: Gibt den Stil der $$arms für diese Schrift an. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
0	Irgendeine
1	Nicht geeignet
2	Gerade $$arms/horizontal
3	Gerade $$arms/wedge
4	Gerade $$arms/vertikal
5	Gerade $$arms/Einzelserife
6	Gerade $$arms/Doppelserife
7	Nicht-gerade $$arms/horizontal
8	Nicht-gerade $$arms/wedge
9	Nicht-gerade $$arms/vertikal
10	Nicht-gerade $$arms/Einzelserife
11	Nicht-gerade $$arms/Doppelserife

bLetterform: Bestimmt die Buchstabenform der Schrift. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
0	Irgendeine
1	Nicht geeignet
2	Normal/Kontakt
3	Normal/gewichtet
4	Normal/umrahmt
5	Normal/abgeflacht
6	Normal/abgerundet
7	Normal/$$off-center
8	Normal/quadratisch
9	Schräg/Kontakt
10	Schräg/gewichtet
11	Schräg/umrahmt
12	Schräg/abgeflacht
13	Schräg/abgerundet
14	Schräg/$$off-center
15	Schräg/quadratisch

bMidline: Bestimmt den Stil der Mittellinie für die Schrift. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
0	Irgendeiner
1	Nicht geeignet
2	Standard/abgeschnitten
3	Standard/punktiert
4	Standard/serifiert
5	Hoch/abgeschnitten
6	Hoch/punktiert
7	Hoch/serifiert
8	Konstant/abgeschnitten
9	Konstant/punktiert
10	Konstant/serifiert
11	Niedrig/abgeschnitten
12	Niedrig/punktiert
13	Niedrig/serifiert

bXHeight: Legt die x-Höhe der Schrift fest. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
0	Irgendeine
1	Nicht geeignet
2	Konstant/klein
3	Konstant/Standard
4	Konstant/groß
5	$$Ducking/klein
6	$$Ducking/Standard
7	$$Ducking/groß

POINTFX 3.1

Definition

```
typedef struct tagPOINTFX { /* fx */
    FIXED x;
    FIXED y;
} POINTFX;
```

Beschreibung Die *POINTFX*-Struktur enthält die Koordinaten von Punkten, die den Umriß eines Zeichens in einer TrueType-Schrift beschreiben. *POINTFX* ist ein Element der Strukuren *TTPOLYCURVE* und *TTPOLYGONHEADER*.

Elemente *x:* Bestimmt die x-Komponente eines Punkts im Umriß eines TrueType-Zeichens.

y: Bestimmt die y-Komponente eines Punkts im Umriß eines TrueType-Zeichens.

Siehe auch *FIXED, TTPOLYCURVE, TTPOLYGONHEADER*

PRINTDLG 3.1

Definition

```
typedef struct tagPRINTDLG { /* pd */
   DWORD lStructSize;
   HWND hwndOwner;
   HGLOBAL hDevMode;
   HGLOBAL hDevNames;
   HDC hDC;
   DWORD Flags;
   UINT nFromPage;
   UINT nToPage;
   UINT nMinPage;
   UINT nMaxPage;
   UINT nCopies;
   HINSTANCE hInstance;
   LPARAM lCustData;
   UINT (CALLBACK* lpfnPrintHook)(HWND, UINT,
         WPARAM, LPARAM);
   UINT (CALLBACK* lpfnSetupHook)(HWND, UINT,
         WPARAM, LPARAM);
   LPCST lpPrintTemplateName;
   LPCST lpSetupTemplateName;
   HGLOBAL hPrintTemplate;
   HGLOBAL hSetupTemplate;
} PRINTDLG;
```

Beschreibung

Die *PRINTDLG*-Struktur enthält Informationen, die das System zur Initialisierung des systemdefinierten Dialogfensters *Drukken* verwendet. Nachdem der Benutzer den *OK*-Schalter gewählt hat, um das Dialogfenster zu schließen, gibt das System Informationen über die Auswahl des Benutzers in dieser Struktur zurück.

Elemente

lStructSize: Gibt die Länge dieser Struktur in Byte an. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

hwndOwner: Bezeichnet das Fenster, zu dem das Dialogfenster gehört. Dieses Element kann jedes gültige Fenster-Handle sein. Es sollte *NULL* sein, wenn das Dialogfenster keinen Besitzer hat.

Wenn das Flag *PD_SHOWHELP* gesetzt ist, muß *hwndOwner* das besitzende Fenster bezeichnen. Die Fensterfunktion dieses Fensters erhält eine Benachrichtigungsbotschaft, wenn der Benutzer den Schalter *Hilfe* wählt. Der Bezeichner der Benachrichtigungsbotschaft ist der Wert, der von der Funktion *RegisterWindowMessage* zurückgegeben wird, wenn ihr *HELPMSGSTRING* als Parameter übergeben wird. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

hDevMode: Bezeichnet ein verschiebbares globales Speicherobjekt, das eine *DEVMODE*-Struktur enthält. Bevor die Funktion *PrintDlg* aufgerufen wird, können die Elemente in dieser Struktur Daten enthalten, die zur Initialisierung der Steuerelemente des Dialogfensters verwendet werden. Wenn die Funktion *PrintDlg* zurückkehrt, bestimmen die Elemente den Status der Steuerelemente.

Wenn die Anwendung die Struktur zur Initialisierung der Steuerelemente des Dialogfensters verwendet, muß sie Speicherplatz reservieren und die *DEVMODE*-Struktur erzeugen. Die Anwendung sollte ein verschiebbares Speicherobjekt reservieren.

Wenn die Anwendung die Struktur nicht zur Initialisierung der Steuerelemente verwendet, kann das Element *hDevMode* den Wert *NULL* haben. In diesem Fall reserviert die Funktion *PrintDlg* Speicher für die Struktur, initialisiert ihre Elemente und liefert ein Handle zurück, um sie zu bezeichnen.

Wenn der Gerätetreiber des angegebenen Druckers keine erweiterten Gerätemodi unterstützt, ist das Element *hDevMode NULL*, wenn die Funktion *PrintDlg* zurückkehrt.

Wenn der Gerätename (im Element *dmDeviceName* der *DEVMODE*-Struktur angegeben) nicht im Abschnitt [devices] der Datei WIN.INI erscheint, gibt die Funktion *PrintDlg* einen Fehlerwert zurück.

Der Wert von *hDevMode* kann sich während der Ausführung der Funktion *PrintDlg* ändern. Dieses Element wird für die Initialisierung des Dialogfensters benötigt und beim Schließen des Fensters gesetzt.

hDevNames: Bezeichnet ein verschiebbares globales Speicherobjekt, das eine *DEVNAMES*-Struktur enthält. Diese Struktur enthält drei Strings, die die Namen des Treibers, des Druckers und der Ausgabeschnittstelle festlegen. Bevor die Funktion *PrintDlg* aufgerufen wird, enthalten die Elemente in dieser Struktur Daten, die zur Initialisierung der Steuerelemente des Dialogfensters verwendet werden. Wenn die Funktion *PrintDlg* zurückkehrt, enthalten die Elemente dieser Struktur die vom Benutzer eingegebenen Strings. Die aufrufende Anwendung verwendet diese Strings, um einen Geräte- oder einen Informationskontext zu erzeugen.

Wenn die Anwendung die Struktur zur Initialisierung der Steuerelemente des Dialogfensters verwendet, muß sie Speicherplatz reservieren und die *DEVMODE*-Struktur erzeugen. Die Anwendung sollte ein verschiebbares globales Speicherobjekt reservieren.

Wenn die Anwendung die Struktur nicht zur Initialisierung der Steuerelemente verwendet, kann das Element *hDevNames NULL* sein. In diesem Fall reserviert die Funktion *PrintDlg* Speicher für die Struktur, initialisiert ihre Elemente (sie ver-

wendet dabei den in der *DEVMODE*-Struktur angegebenen Druckernamen) und liefert ein Handle zurück, das die Struktur bezeichnet. Wenn die Funktion *PrintDlg* die Elemente der *DEVNAMES*-Struktur initialisiert, verwendet sie den ersten Schnittstellennamen, der im Abschnitt [devices] der Datei WIN.INI erscheint. So verwendet die Funktion z.B. »LPT1« als Schnittstellennamen, wenn der folgende String im Abschnitt [devices] erscheint:

```
PCL/HP LaserJet=HPPCL,LPT1:,LPT2:
```

Wenn die Elemente *hDevMode* und *hDevNames NULL* sind, bestimmt *PrintDlg* den aktuellen voreingestellten Drucker für *hDevNames*.

Der Wert von *hDevNames* kann sich während der Ausführung der Funktion *PrintDlg* ändern. Dieses Element wird für die Initialisierung des Dialogfensters benötigt und beim Schließen des Fensters gesetzt.

hDC: Bezeichnet entweder einen Geräte- oder einen Informationskontext, abhängig davon, ob das Element *Flags* das Flag *PD_RETURNDC* oder *PC_RETURNIC* enthält. Wenn keines der beiden Flags angegeben wird, ist der Wert dieses Elements nicht definiert. Wenn beide angegeben werden, hat *hDC* den Wert *PD_RETURNDC*.

Dieses Element wird beim Schließen des Fensters gesetzt.

Flags: Legt die Initialisierungs-Flags des Dialogfensters fest. Das Element kann eine Kombination der folgenden Werte sein:

Wert	Bedeutung
PD_ALLPAGES	Das Schaltfeld *Alles* war ausgewählt, als der Benutzer das Dialogfenster geschlossen hat. Dieser Wert wird als Platzhalter benutzt, um anzuzeigen, daß die Flags *PD_PAGENUMS* und *PD_SELECTION* nicht gesetzt sind. Dieser Wert wird für die Initialisierung des Dialogfensters benötigt und kann beim Schließen des Fensters gesetzt werden.
PD_COLLATE	Das Auswahlfeld *Kopien* wird bei der Erzeugung des Dialogfensters markiert. Wenn die Funktion *PrintDlg* zurückkehrt, zeigt dieses Flag den Status an, in dem der Benutzer das Auswahlfeld verlassen hat. Dieses Flag wird für die Initialisierung des Dialogfensters benötigt und kann beim Schließen des Fensters gesetzt werden.

Wert	Bedeutung
PD_DISABLEPRINT-TOFILE	Deaktiviert das Auswahlfeld *Ausdruck in Datei*.
PD_ENABLEPRINT-HOOK	Aktiviert die im Element *lpfnPrintHook* dieser Struktur angegebene Filterfunktion.
PD_ENABLEPRINT-TEMPLATE	Veranlaßt das System, die von den Elementen *hInstance* und *lpPrintTemplate-Name* bezeichnete Dialogfensterschablone zur Erzeugung des Dialogfensters zu verwenden.
PD_ENABLEPRINT-TEMPLATEHANDLE	Weist darauf hin, daß das Element *hPrint-Template* einen Datenblock bezeichnet, der eine bereits geladene Dialogfensterschablone enthält. Das System ignoriert das Element *hInstance*, wenn dieses Flag angegeben wird.
PD_ENABLESETUP-HOOK	Aktiviert die im Element *lpfnSetupHook* dieser Struktur angegebene Filterfunktion.
PD_ENABLESETUP-TEMPLATE	Veranlaßt das System, die von den Elementen *hInstance* und *lpSetupTemplate-Name* bezeichnete Dialogfensterschablone zur Erzeugung des Dialogfensters *Drukker einrichten* zu verwenden.
PD_ENABLESETUP-TEMPLATEHANDLE	Das Element *hSetupTemplate* bezeichnet einen Datenblock, der eine bereits geladene Dialogfensterschablone enthält. Das System ignoriert das Element *hInstance*, wenn dieses Flag angegeben wird.
PD_HIDEPRINTTOFILE	Verbirgt und deaktiviert das Auswahlfeld *Ausdruck in Datei*.
PD_NOPAGENUMS	Deaktiviert das Schaltfeld *Seiten* und die zugeordneten Editierfelder.
PD_NOSELECTION	Deaktiviert das Schaltfeld *Markierung*.
PD_NOWARNING	Verhindert, daß die Warnmeldung ausgegeben wird, wenn es keinen voreingestellten Drucker gibt.
PD_PAGENUMS	Das Schaltfeld *Seiten* ist bei der Erzeugung des Dialogfensters ausgewählt. Sobald die Funktion *PrintDlg* zurückkehrt, wird dieses Flag gesetzt, wenn der Schalter *Seiten* im ausgewählten Zustand ist.

Wert	Bedeutung
	Wenn weder *PD_PAGENUMS* noch *PD_SELECTION* gesetzt ist, ist das Schaltfeld *Alles* ausgewählt. Dieses Flag wird für die Initialisierung des Dialogfensters benötigt und kann beim Schließen des Fensters gesetzt werden.
PD_PRINTSETUP	Veranlaßt das System, das Dialogfenster *Drucker einrichten* anstelle des Fensters *Drucken* anzuzeigen.
PD_PRINTTOFILE	Das Auswahlfeld *Ausdruck in Datei* wird bei der Erzeugung des Dialogfensters ausgewählt. Dieses Flag wird für die Initialisierung des Dialogfensters benötigt und kann beim Schließen des Fensters gesetzt werden.
PD_RETURNDC	Veranlaßt die Funktion *PrintDlg*, einen Gerätekontext zurückzugeben, der den Einstellungen des Benutzers im Dialogfenster entspricht. Das Handle des Gerätekontexts wird im Element *hDC* zurückgeliefert. Wenn weder *PD_RETURNDC* noch *PD_RETURNIC* angegeben wird, ist der Parameter *hDC* bei der Ausgabe nicht definiert.
PD_RETURNDEFAULT	Veranlaßt die Funktion *PrintDlg*, *DEVMODE*- und *DEVNAMES*-Strukturen zurückzuliefern, die für den voreingestellten Drucker des Systems initialisiert sind. *PrintDlg* tut dies, ohne ein Dialogfenster anzuzeigen. Sowohl das Element *hDevNames* als auch *hDevMode* sollten *NULL* sein; andernfalls gibt die Funktion einen Fehlerwert zurück. Wenn der voreingestellte Drucker von einem alten Druckertreiber unterstützt wird (früher als Windows 3.0), wird nur das Element *hDevNames* zurückgegeben – *hDevMode* ist dann *NULL*.
PD_RETURNIC	Veranlaßt die Funktion *PrintDlg*, einen Informationskontext zurückzugeben, der den Einstellungen des Benutzers im Dialogfenster entspricht. Er wird im Element *hDC* zurückgeliefert.

Wert	Bedeutung
	Wenn weder *PD_RETURNDC* noch *PD_RETURNIC* angegeben wird, ist der Parameter *hDC* bei der Ausgabe von Informationen nicht definiert.
PD_SELECTION	Das Schaltfeld *Markierung* wird bei der Erzeugung des Dialogfensters ausgewählt. Sobald die Funktion *PrintDlg* zurückkehrt, wird dieses Flag gesetzt, wenn der Schalter im ausgewählten Zustand ist. Wenn weder *PD_PAGENUMS* noch *PD_SELECTION* gesetzt ist, ist das Schaltfeld *Alles* ausgewählt. Dieses Flag wird für die Initialisierung des Dialogfensters benötigt und kann beim Schließen des Fensters gesetzt werden.
PD_SHOWHELP	Veranlaßt das Dialogfenster, den *Hilfe*-Schalter anzuzeigen. Wenn dieses Flag angegeben wird, darf *hwndOwner* nicht *NULL* sein.
PD_USEDEVMODE-COPIES	Deaktiviert das Editierfeld *Kopien*, wenn ein Druckertreiber den Druck mehrerer Kopien nicht unterstützt. Wenn ein Treiber mehrere Kopien unterstützt, zeigt das Setzen dieses Flag an, daß die Funktion *PrintDlg* die gewünschte Anzahl von Kopien im Element *dmCopies* der *DEVMODE*-Struktur und den Wert 1 im Element *nCopies* der *PRINTDLG*-Struktur speichern soll. Wenn dieses Flag nicht gesetzt ist, speichert die Struktur *PRINTDLG* den Wert 1 im Element *dmCopies* der *DEVMODE*-Struktur und die gewünschte Anzahl von Kopien im Element *nCopies* der *PRINTDLG*-Struktur.

Diese Flags können – wenn nicht anders angegeben – bei der Initialisierung der Struktur gesetzt werden.

nFromPage: Bestimmt den anfänglichen Wert für die Startseite im Editierfeld *Von*. Wenn die Funktion *PrintDlg* beendet ist, gibt dieses Element die Seite an, mit der das Drucken beginnen soll. Dieser Wert ist nur gültig, wenn das Flag *PD_PAGENUMS* gesetzt ist. Der Höchstwert für dieses Element ist 0xFFFE. Wenn 0xFFFF angegeben wird, bleibt das Editierfeld *Von* leer. Dieses Element wird für die Initialisierung des Dialogfensters benötigt und beim Schließen des Fensters gesetzt.

nToPage: Bestimmt den anfänglichen Wert für die letzte Seite im Editierfeld *Bis*. Wenn die Funktion *PrintDlg* zurückkehrt, gibt dieses Element die letzte Seite an, die gedruckt werden soll. Dieser Wert ist nur gültig, wenn das Flag *PD_PAGENUMS* gesetzt ist. Der Höchstwert für dieses Element ist 0xFFFE. Wenn 0xFFFF angegeben wird, bleibt das Editierfeld *Bis* leer. Dieses Element wird für die Initialisierung des Dialogfensters benötigt und beim Schließen des Fensters gesetzt.

nMinPage: Bestimmt die kleinste Seitenzahl, die in den Editierfeldern *Von* und *Bis* festgelegt werden kann. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

nMaxPage: Bestimmt die größte Seitenzahl, die in den Editierfeldern *Von* und *Bis* festgelegt werden kann. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

nCopies: Vor dem Aufruf der Funktion *PrintDlg* bestimmt dieses Element den Wert, der zur Initialisierung des Editierfelds *Kopien* verwendet wird, wenn das Element *hDevMode* den Wert *NULL* hat; andernfalls enthält das Element *dmCopies* der *DEVMODE*-Struktur den Wert, der zur Initialisierung des Editierfelds *Kopien* verwendet wird.

Wenn die Funktion *PrintDlg* zurückkehrt, ist der von diesem Element festgelegte Wert von der Windows-Version abhängig, für die der Druckertreiber geschrieben worden ist. Bei Druckertreibern, die für Windows-Versionen vor 3.0 geschrieben wurden, bestimmt dieses Element die Anzahl der vom Benutzer gewünschten Kopien, wenn das Flag *PD_USEDEVMODECOPIES* nicht gesetzt wurde; andernfalls enthält es den Wert 1 und die tatsächliche Anzahl der gewünschten Kopien erscheint in der *DEVMODE*-Struktur.

Dieses Element wird für die Initialisierung des Dialogfensters benötigt und beim Schließen des Fensters gesetzt.

hInstance: Bezeichnet einen Datenblock, der die vom Element *lpPrintTemplateName* oder *lpSetupTemplateName* angegebene, bereits geladene Dialogfensterschablone enthält. Dieses Element wird nur dann verwendet, wenn das Element *Flags* das Flag *PD_ENABLEPRINTTEMPLATE* oder *PD_ENABLESETUPTEMPLATE* enthält. Andernfalls wird es ignoriert. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

lCustData: Gibt anwendungsdefinierte Daten an, die das System an die durch das Element *lpfnPrintHook* oder *lpfnSetupHook* bezeichnete Filterfunktion übergibt. Das System übergibt einen Zeiger auf die *PRINTDLG*-Struktur im Parameter *lParam* der *WM_INITDIALOG*-Botschaft. Dieser Zeiger kann verwendet werden, um das Element *lCustData* abzurufen.

lpfnPrintHook: Zeigt auf die exportierte Filterfunktion, die Dialogfensterbotschaften bearbeitet, wenn die Anwendung das Dialogfenster *Drucken* anpaßt. Dieses Element wird ignoriert, wenn das Flag *PD_ENABLEPRINTHOOK* im Element *Flags* nicht angegeben wird. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

lpfnSetupHook: Zeigt auf die exportierte Filterfunktion, die Dialogfensterbotschaften bearbeitet, wenn die Anwendung das Dialogfenster *Drucker einrichten* anpaßt. Dieses Element wird ignoriert, wenn das Flag *PD_ENABLESETUPHOOK* im Element *Flags* nicht angegeben wird. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

lpPrintTemplateName: Zeigt auf einen nullterminierten String, der die Dialogfensterschablone bestimmt, die anstelle der Standard-Dialogfensterschablone in COMMDLG.DLL verwendet werden soll. Eine Anwendung muß die Konstante *PD_ENABLEPRINTTEMPLATE* im Element *Flags* angeben, um die Filterfunktion zu aktivieren; andernfalls ignoriert das System dieses Strukturelement. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

lpSetupTemplateName: Zeigt auf einen nullterminierten String, der die Dialogfensterschablone bestimmt, die anstelle der Standard-Dialogfensterschablone in COMMDLG.DLL verwendet werden soll. Eine Anwendung muß die Konstante *PD_ENABLESETUPTEMPLATE* im Element *Flags* angeben, um die Filterfunktion zu aktivieren; andernfalls ignoriert das System dieses Strukturelement. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

hPrintTemplate: Bezeichnet das Handle des globalen Speicherobjekts, das die bereits geladene Dialogfensterschablone enthält, die anstelle der voreingestellten Schablone in COMMDLG.DLL für das Dialogfenster *Drucken* verwendet werden soll. Um die Dialogfensterschablone verwenden zu können, muß das Flag *PD_ENABLEPRINTTEMPLATEHANDLE* gesetzt werden. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

hSetupTemplate: Bezeichnet das Handle des globalen Speicherobjekts, das die bereits geladene Dialogfensterschablone enthält, die anstelle der voreingestellten Schablone in COMMDLG.DLL für das Dialogfenster *Drucker einrichten* verwendet werden soll. Um die Dialogfensterschablone verwenden zu können, muß das Flag *PD_ENABLESETUPTEMPLATEHANDLE* gesetzt werden. Dieses Element wird für die Initialisierung des Dialogfensters benötigt.

Siehe auch *CreateDC, CreateIC, PrintDlg, DEVMODE, DEVNAMES*

RASTERIZER_STATUS 3.1

Definition

```
typedef struct tagRASTERIZER_STATUS { /* rs */
   int nSize;
   int wFlags;
   int nLanguageID;
} RASTERIZER_STATUS;
```

Beschreibung Die *RASTERIZER_STATUS*-Struktur enthält Informationen darüber, ob TrueType installiert ist. Diese Strukur wird gefüllt, wenn eine Anwendung die Funktion *GetRasterizerCaps* aufruft.

Elemente *nSize:* Gibt die Größe der *RASTERIZER_STATUS*-Struktur in Byte an.

wFlags: Gibt an, ob zumindest eine TrueType-Schriftart installiert und ob TrueType aktiviert ist. Dieser Wert ist *TT_AVAILABLE* und/oder *TT_ENABLED*, wenn TrueType auf dem System vorhanden ist.

nLanguageID: Bestimmt die Sprache in der SETUP.INF-Datei des Systems. Weitere Informationen über Microsoft-Sprachenbezeichner finden Sie in der Beschreibung der *STRINGTABLE*-Struktur.

Siehe auch *GetRasterizerCaps*

SEGINFO 3.1

Definition

```
typedef struct tagSEGINFO {
   UINT offSegment;
   UINT cbSegment;
   UINT flags;
   UINT cbAlloc;
   HGLOBAL h;
   UINT alignShift;
   UINT reserved[2];
} SEGINFO;
```

Beschreibung Die *SEGINFO*-Struktur enthält Informationen über ein Daten- oder ein Codesegment. Diese Struktur wird von der Funktion *GetCodeInfo* gefüllt.

Elemente *offSegment:* Gibt den Offset (in Sektoren) zum Inhalt der Segmentdaten relativ zum Anfang der Datei an. Null bedeutet, daß keine Dateidaten verfügbar sind. Die Größe des Sektors wird ermittelt, indem der im Element *alignShift* angegebene Wert um ein Bit nach links verschoben wird.

cbSegment: Gibt die Länge des Segments in der Datei in Byte an. Null bedeutet 64 KByte.

flags: Enthält Flags, die Segmentattribute festlegen. Die folgende Liste beschreibt diese Flags:

Bit	Bedeutung
0-2	Bestimmt den Segmenttyp. Wenn Bit 0 auf 1 gesetzt ist, ist das Segment ein Datensegment. Andernfalls ist es ein Codesegment.
3	Legt fest, ob Segmentdaten iteriert werden. Wenn dieses Bit auf 1 gesetzt ist, ist dies der Fall.
4	Bestimmt, ob das Segment verschiebbar oder fest ist. Wenn dieses Bit auf 1 gesetzt ist, ist das Segment verschiebbar. Andernfalls ist es fest.
5-6	Reserviert.
7	Legt fest, ob das Segment ein Nur-Lesen-Datensegment oder ein Nur-Ausführen-Codesegment ist. Wenn dieses Bit auf 1 gesetzt und das Segment ein Codesegment ist, ist es ein Nur-Ausführen-Segment. Wenn es auf 0 gesetzt und das Segment ein Datensegment ist, ist es ein Nur-Lesen-Segment.
8	Gibt an, ob das Segment zugeordnete Relozierungsinformationen hat. Wenn dieses Bit auf 1 gesetzt ist, hat das Segment Relozierungsinformationen. Andernfalls hat es keine.
9	Gibt an, ob das Segment Debug-Informationen enthält. Wenn dieses Bit auf 1 gesetzt ist, hat das Segment Debug-Informationen. Andernfalls hat es keine.
10-15	Reserviert.

cbAlloc: Gibt die Gesamtgröße des für das Segment reservierten Speichers an. Dieser Wert kann größer als die tatsächliche Größe des Segments sein. Null bedeutet 64 KByte.

h: Bezeichnet den globalen Speicher für das Segment.

alignShift: Gibt die Größe des adressierbaren Sektors als Exponent von 2 an. Eine ausführbare Datei füllt das Code-, Daten- und Ressourcensegment der Anwendung mit Null-Byte auf, so daß die Segmente immer ein Vielfaches der Dateisegmentgröße sind. Windows verwirft die zusätzlichen Byte, wenn es die Segmente aus der Datei lädt.

reserved: Gibt zwei reservierte Werte vom Typ *UINT* an.

Siehe auch *GetCodeInfo*

SIZE 3.1

Definition

```
typedef struct tagSIZE {
   int cx;
   int cy;
} SIZE;
```

Beschreibung Die *SIZE*-Struktur enthält Grafikbildschirm-, Fenster-, Text- und Bitmap-Abmessungen und den Filter des xy-Seitenverhältnisses für einige erweiterte Funktionen von Windows 3.1.

Elemente *cx:* Bestimmt die x-Abmessung, wenn die Funktion zurückkehrt.

cy: Bestimmt die y-Abmessung, wenn die Funktion zurückkehrt.

Siehe auch *GetAspectRatioFilterEx, GetBitmapDimensionEx, GetTextExtentPoint, GetViewportExtEx, GetWindowExtEx, ScaleViewportExtEx, ScaleWindowExtEx, SetBitmapDimensionEx, SetViewportExtEx, SetWindowExtEx*

STACKTRACEENTRY 3.1

Definition

```
#include <toolhelp.h>

typedef struct tagSTACKTRACEENTRY { /* ste */
   DWORD dwSize;
   HTASK hTask;
   WORD wSS;
   WORD wBP;
   WORD wCS;
   WORD wIP;
   HMODULE hModule;
   WORD wSegment;
   WORD wFlags;
} STACKTRACEENTRY;
```

Beschreibung Die *STACKTRACEENTRY*-Struktur enthält Informationen über einen Stack-Frame. Diese Informationen ermöglichen einer Anwendung, den Stack einer bestimmten Task zurückzuverfolgen.

Elemente *dwSize:* Gibt die Größe der *STACKTRACEENTRY*-Struktur in Byte an.

hTask: Bezeichnet das Task-Handle für den Stack.

wSS: Enthält den Wert des SS-Registers. Dieser Wert wird mit dem des Elements *wBP* verwendet, um den nächsten Eintrag in der Stack-Ablaufverfolgungstabelle zu ermitteln.

wBP: Enthält den Wert des BP-Registers. Dieser Wert wird mit dem des Elements *wSS* verwendet, um den nächsten Eintrag in der Stack-Ablaufverfolgungstabelle zu ermitteln.

wCS: Enthält den Wert des CS-Registers beim Rücksprung. Dieser Wert wird mit dem des Elements *wIP* verwendet, um den Rückgabewert der Funktion zu bestimmen.

wIP: Enthält den Wert des IP-Registers beim Rücksprung. Dieser Wert wird mit dem des Elements *wCS* verwendet, um den Rückgabewert der Funktion zu bestimmen.

hModule: Bezeichnet das Modul, das die aktuell ausgeführte Funktion enthält.

wSegment: Enthält die Segmentnummer des aktuellen Selektors.

wFlags: Zeigt den Frame-Typ an. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
FRAME_FAR	Das CS-Register enthält ein gültiges Codesegment.
FRAME_NEAR	Das CS-Register ist Null.

Siehe auch *StackTraceCSIPFirst, StackTraceNext, StackTraceFirst*

SYSHEAPINFO 3.1

Definition

```
#include <toolhelp.h>

typedef struct tagSYSHEAPINFO { /* shi */
   DWORD dwSize;
   DWORD wUserFreePercent;
   DWORD wGDIFreePercent;
   HGLOBAL hUserSegment;
   HGLOBAL hGDISegment;
} SYSHEAPINFO;
```

Beschreibung Die *SYSHEAPINFO*-Struktur enthält Informationen über die USER- und GDI-Module.

Elemente *dwSize:* Gibt die Größe der *SYSHEAPINFO*-Struktur in Byte an.

wUserFreePercent: Gibt an, wieviel Prozent des lokalen USER-Heap frei sind.

wGDIFreePercent: Gibt an, wieviel Prozent des lokalen GDI-Heap frei sind.

hUserSegment: Bezeichnet das DGROUP-Segment des lokalen USER-Heap.

hGDISegment: Bezeichnet das DGROUP-Segment des lokalen GDI-Heap.

Siehe auch *SystemHeapInfo*

TASKENTRY 3.1

Definition

```
#include <toolhelp.h>

typedef struct tagTASKENTRY { /* te */
   DWORD dwSize;
   HTASK hTask;
   HTASK hTaskParent;
   HINSTANCE hInst;
   HMODULE hModule;
   WORD wSS;
   WORD wSP;
   WORD wStackTop;
   WORD wStackMinimum;
   WORD wStackBottom;
   WORD wcEvents;
   HGLOBAL hQueue;
   char szModule[MAX_MODULE_NAME + 1];
   WORD wPSPOffset;
   HANDLE hNext;
} TASKENTRY;
```

Beschreibung Die *TASKENTRY*-Struktur enthält Informationen über eine Task.

Elemente *dwSize:* Gibt die Größe der *TASKENTRY*-Struktur in Byte an.

hTask: Bezeichnet das Task-Handle für den Stack.

hTaskParent: Bezeichnet das Handle der übergeordneten Task.

hInst: Bezeichnet das Instanz-Handle der Task. Dieser Wert ist mit dem DGROUP-Segmentselektor der Task identisch.

hModule: Bezeichnet das Modul, das die aktuell ausgeführte Funktion enthält.

wSS: Enthält den Wert des SS-Registers.

wSP: Enthält den Wert des SP-Registers.

wStackTop: Gibt den Offset zur Spitze des Stack (niedrigste Adresse auf dem Stack) an.

wStackMinimum: Gibt die niedrigste Segmentnummer des Stack während der Ausführung einer Task an.

wStackBottom: Gibt den Offset zum Ende des Stack (höchste Adresse auf dem Stack) an.

wcEvents: Gibt die Anzahl der vorliegenden Ereignisse an.

hQueue: Bezeichnet die Task-Warteschlange.

szModule: Gibt den Namen des Moduls an, das die aktuell ausgeführte Funktion enthält.

wPSPOffset: Bestimmt den Offset vom Präfix des Programmsegments (PSP) zum Anfang des ausführbaren Codesegments.

hNext: Bezeichnet den nächsten Eintrag in der Task-Liste. Dieses Element ist für die interne Verwendung durch Windows reserviert.

Siehe auch *TaskFindHandle, TaskFirst, TaskNext*

TIMERINFO 3.1

Definition

```
#include <toolhelp.h>

typedef struct tagTIMERINFO { /* ti */
   DWORD dwSize;
   DWORD dwmsSinceStart;
   DWORD dwmsThisVM;
} TIMERINFO;
```

Beschreibung Die *TIMERINFO*-Struktur enthält die Zeit, die verstrichen ist, seit die aktuelle Task aktiv wurde und seit die virtuelle Maschine (VM) gestartet ist.

Elemente *dwSize:* Gibt die Größe der *TIMERINFO*-Struktur in Byte an.

dwmsSinceStart: Enthält die Anzahl der Millisekunden, seit die aktuelle Task aktiv wurde.

dwmsThisVM: Enthält die Anzahl der Millisekunden, seit die virtuelle Maschine (VM) gestartet wurde.

Hinweis Im Standardmodus sind die Werte für *dwmsSinceStart* und *dwmsThisVM* identisch.

Siehe auch *TimerCount*

TTPOLYCURVE 3.1

Definition

```
typedef struct tagTTPOLYCURVE {
   UINT wType;
   UINT cpfx;
   POINTFX apfx[1];
} TTPOLYCURVE;
```

Beschreibung Die TTPOLYCURVE-Struktur enthält Informationen über eine Kurve im Umriß eines TrueType-Zeichens.

Elemente *wType:* Gibt den von der Struktur beschriebenen Kurventyp an. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
TT_PRIM_LINE	Die Kurve ist ein Linienzug.
TT_PRIM_QSPLINE	Die Kurve ist eine quadratische Spline.

cpfx: Gibt die Anzahl der *POINTFX*-Strukturen im Array an.

apfx: Gibt ein Array von *POINTFX*-Strukturen an, die den Linienzug oder die quadratische Spline definieren.

Hinweis Wenn eine Anwendung die Funktion *GetGlyphOutline* aufruft, wird der Umriß eines Zeichenkörpers für ein TrueType-Zeichen in einer *TTPOLYGONHEADER*-Struktur zurückgegeben, gefolgt von so vielen *TTPOLYCURVE*-Strukturen, wie für die Beschreibung des Zeichenkörpers benötigt werden.

Alle Punkte werden in *POINTFX*-Strukturen zurückgegeben und stellen absolute Positionen dar, nicht relative Verschiebungen. Der im Element *pfxStart* der *TTPOLYGONHEADER*-Struktur angegebene Anfangspunkt ist der Punkt, an dem die Umrißlinie für eine Kontur beginnt. Die folgenden *TTPOLYCURVE*-Strukturen können entweder Polyline- oder Spline-Records sein.

Polyline-Records sind eine Reihe von Punkten; Linien, die zwischen den Punkten gezogen werden, beschreiben den Umriß des Zeichens. Spline-Records repräsentieren quadratische Kurven, die von TrueType verwendet werden (d.h. quadratische b-Splines).

Siehe auch *POINTFX, TTPOLYGONHEADER*

TTPOLYGONHEADER 3.1

Definition
```
typedef struct tagTTPOLYGONHEADER {
   DWORD cb;
   DWORD dwType;
   POINTFX pfxStart;
} TTPOLYGONHEADER;
```

Beschreibung Die *TTPOLYGONHEADER*-Struktur legt die Startposition und den Typ eines TrueType-Zeichenumrisses fest.

Elemente *cb:* Gibt die von der *TTPOLYGONHEADER*-Struktur benötigte Anzahl Byte an.

dwType: Bestimmt den Typ des Zeichenumrisses der zurückgegeben wird. Derzeit muß dieser Wert *TT_POLYGON_TYPE* sein.

pfxStart: Legt die Startposition des Zeichenumrisses fest.

Hinweis Der Zeichenumriß wird von mehreren *TTPOLYCURVE*-Strukturen beschrieben, die der *TTPOLYGONHEADER*-Struktur folgen.

Siehe auch *POINTFX, TTPOLYCURVE*

VS_FIXEDFILEINFO 3.1

Definition

```
typedef struct tagVS_FIXEDFILEINFO { /* vsffi         */
  DWORD dwSignature;        /* z.B. 0xFEEF04BD       */
  DWORD dwStrucVersion;     /* z.B. 0x00000042="0.42" */
  DWORD dwFileVersionMS;    /* z.B. 0x00030075="3.75" */
  DWORD dwFileVersionLS;    /* z.B. 0x00000031="0.31" */
  DWORD dwProductVersionMS; /* z.B. 0x00030010="3.10" */
  DWORD dwProductVersionLS; /* z.B. 0x00000031="0.31" */
  DWORD dwFileFlagsMask;    /* = 0x3F für Version
                               "0.42"                */
  DWORD dwFileFlags;        /* z.B. vff_Debug |
                               vff_Prerelease        */
  DWORD dwFileOS;           /* z.B. vos_DOS_Windows16 */
  DWORD dwFileType;         /* z.B. vft_DRIVER       */
  DWORD dwFileSubType;      /* z.B. vft_DRV_Keyboard  */
  DWORD dwFileDateMS;       /* z.B. 0                */
  DWORD dwFileDateLS;       /* z.B. 0                */
} FIXED;
```

Beschreibung Die *VS_FIXEDFILEINFO*-Struktur enthält Versionsinformationen über eine Datei.

Elemente *dwSignature:* Legt den Wert 0xFEEF04BD fest.

dwStrucVersion: Gibt die binäre Versionsnummer dieser Struktur an. Das höherwertige *WORD* enthält die Hauptversionsnummer und das niederwertige die Nebenversionsnummer. Dieser Wert muß größer als 0x00000029 sein.

dwFileVersionMS: Gibt die höherwertigen 32 Bits der binären Versionsnummer der Datei an. Der Wert dieses Elements wird zusammen mit dem Wert von *dwFileVersionLS* verwendet, um eine 64-Bit-Versionsnummer zu bilden.

dwFileVersionLS: Gibt die niederwertigen 32 Bits der binären Versionsnummer der Datei an. Der Wert dieses Elements wird zusammen mit dem Wert von *dwFileVersionMS* verwendet, um eine 64-Bit-Versionsnummer zu bilden.

dwProductVersionMS: Gibt die höherwertigen 32 Bits der binären Versionsnummer des Produkts an, mit dem diese Datei ausgeliefert wurde. Der Wert dieses Elements wird zusammen mit dem Wert von *dwProductVersionLS* verwendet, um eine 64-Bit-Versionsnummer zu bilden.

dwProductVersionLS: Gibt die niederwertigen 32 Bits der binären Versionsnummer des Produkts an, mit dem diese Datei ausgeliefert wurde. Der Wert dieses Elements wird zusammen mit dem Wert von *dwProductVersionMS* verwendet, um eine 64-Bit-Versionsnummer zu bilden.

dwFileFlagsMask: Bestimmt, welche Bits im Element *dwFileFlags* gültig sind. Wenn ein Bit gesetzt wird, ist das entsprechende Bit im Element *dwFileFlags* gültig.

dwFileFlags: Legt die booleschen Attribute der Datei fest. Die Attribute können eine Kombination der folgenden Werte sein:

Wert	Bedeutung
VS_FF_DEBUG	Die Datei enthält Debug-Informationen oder ist mit aktivierter Debug-Option compiliert worden.
VS_FF_INFOINFERRED	Die Datei enthält eine dynamisch erzeugte Versionsinformationsressource. Einige Blöcke der Ressource können leer oder falsch sein. Dieser Wert ist nicht für Versionsinformationsressourcen bestimmt, die durch die Anweisung *VERSIONINFO* erzeugt wurden.
VS_FF_PATCHED	Die Datei wurde geändert und ist nicht mit der ausgelieferten Originaldatei dieser Versionsnummer identisch.
VS_FF_PRERELEASE	Die Datei ist eine Entwicklungsversion, nicht ein für den Handel freigegebenes Produkt.
VS_FF_PRIVATEBUILD	Die Datei wurde nicht unter Verwendung der Standardfunktionen erzeugt. Wenn dieser Wert angegeben wird, muß der *StringFileInfo*-Block einen *PrivateBuild*-String enthalten.
VS_FF_SPECIALBUILD	Die Datei wurde von der Originalfirma unter Verwendung der Standardfunktionen erzeugt, ist aber eine Variation der Standarddatei der gleichen Versionsnummer. Wenn dieser Wert angegeben wird, muß der *StringFileInfo*-Block einen *SpecialBuild*-String enthalten.

dwFileOS: Gibt das Betriebssystem an, für das die Datei entworfen wurde. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
VOS_UNKNOWN	Das Betriebssystem, für das die Datei entworfen wurde, ist Windows nicht bekannt.
VOS_DOS	Datei wurde für MS-DOS entworfen.
VOS_NT	Datei wurde für Windows NT entworfen.
VOS_WINDOWS16	Datei wurde für Windows Version 3.0 oder später entworfen.
VOS_WINDOWS32	Datei wurde für 32-Bit-Windows entworfen.
VOS_DOS_WINDOWS16	Datei wurde für Windows Version 3.0 oder später entworfen, das mit MS-DOS läuft.
VOS_DOS_WINDOWS32	Datei wurde für 32-Bit-Windows entworfen, das mit MS-DOS läuft.
VOS_NT_-WINDOWS32	Datei wurde für 32-Bit-Windows entworfen, das mit Windows NT läuft.

Die Werte 0x00002L, 0x00003L, 0x20000L und 0x30000L sind reserviert.

dwFileType: Gibt den allgemeinen Dateityp an. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
VFT_UNKNOWN	Der Dateityp ist Windows nicht bekannt.
VFT_APP	Die Datei enthält eine Anwendung.
VFT_DLL	Die Datei enthält eine dynamische Linkbibliothek (DLL).
VFT_DRV	Die Datei enthält einen Gerätetreiber. Wenn das Element *dwFileType* diesen Wert hat, enthält *dwFileSubType* eine genauere Beschreibung des Treibers.
VFT_FONT	Die Datei enthält eine Schrift. Wenn das Element *dwFileType* diesen Wert hat, enthält *dwFileSubType* eine genauere Beschreibung der Schrift.
VFT_VXD	Die Datei enthält ein virtuelles Gerät.
VFT_STATIC_LIB	Die Datei enthält eine statische Linkbibliothek.

Alle anderen Werte sind für die Verwendung durch Microsoft reserviert.

dwFileSubType: Gibt die Funktion der Datei an. Dieses Element ist Null, wenn das Element *dwFileType* nicht den Wert *VFT_DRV*, *VFT_FONT* oder *VFT_VXD* hat.

Wenn *dwFileType VFT_DRV* ist, kann *dwFileSubtype* einen der folgenden Werte annehmen:

Wert	Bedeutung
VFT2_UNKNOWN	Der Treibertyp ist Windows nicht bekannt.
VFT2_DRV_COMM	Die Datei enthält einen seriellen Treiber.
VFT2_DRV_PRINTER	Die Datei enthält einen Druckertreiber.
VFT2_DRV_KEYBOARD	Die Datei enthält einen Tastaturtreiber.
VFT2_DRV_LANGUAGE	Die Datei enthält einen Sprachentreiber.
VFT2_DRV_DISPLAY	Die Datei enthält einen Bildschirmtreiber.
VFT2_DRV_MOUSE	Die Datei enthält einen Maustreiber.
VFT2_DRV_NETWORK	Die Datei enthält einen Netztreiber.
VFT2_DRV_SYSTEM	Die Datei enthält einen Systemtreiber.
VFT2_DRV_INSTALLABLE	Die Datei enthält einen installierbaren Treiber.
VFT2_DRV_SOUND	Die Datei enthält einen Klangtreiber.

Wenn *dwFileType VFT_FONT* ist, dann kann *dwFileSubType* einen der folgenden Werte annehmen:

Wert	Bedeutung
VFT2_UNKNOWN	Der Schrifttyp ist Windows nicht bekannt.
VFT2_FONT_RASTER	Die Datei enthält eine Rasterschrift.
VFT2_FONT_VECTOR	Die Datei enthält eine Vektorschrift.
VFT2_FONT_TRUETYPE	Die Datei enthält eine TrueType-Schrift.

Wenn *dwFileType* den Wert *VFT_VXD* hat, dann enthält *dwFileSubType* den Bezeichner des virtuellen Geräts, der im Steuerblock (DCB) des virtuellen Geräts enthalten ist.

Alle hier nicht aufgelisteten Werte für *dwFileSubType* sind für die Verwendung durch Microsoft reserviert.

dwFileDateMS: Gibt die höherwertigen 32 Bits einer binären Datum/Zeit-Angabe der Datei an. Der Wert dieses Elements

wird mit dem von *dwFileDateLS* verwendet, um eine 64-Bit-Zahl zu bilden, die Erstellungsdatum und -zeit der Datei angibt.

dwFileDateLS: Gibt die niederwertigen 32 Bits einer binären Datum/Zeit-Angabe der Datei an. Der Wert dieses Elements wird mit dem von *dwFileDateMS* verwendet, um eine 64-Bit-Zahl zu bilden, die Erstellungsdatum und -zeit der Datei angibt.

Hinweis

Die in dieser Struktur angegebenen binären Versionsnummern sollten ganze Zahlen und keine Zeichen-Strings sein. Bei einer Datei oder einem Produkt, das Dezimalpunkte oder Buchstaben in seiner Versionsnummer hat, sollte die entsprechende binäre Versionsnummer eine angemessene numerische Darstellung sein.

Ein Drittentwickler kann diese Dateiversionswerte verwenden, um ein privates Versionsnumerierungsschema wiederzugeben. Jede neue Version des Produkts hat eine höhere Nummer als die vorhergehende Version. Die Bibliothek für die Installation von Dateien verwendet diese Werte, wenn sie das Alter von Dateien vergleicht.

Der Ressourcen-Compiler von Windows setzt die Elemente *dwFileDateMS* und *dwFileDateLS* auf Null.

Siehe auch *VerQueryValue*

WINDEBUGINFO 3.1

Definition

```
typedef struct tagWINDEBUGINFO {
   UINT flags;
   DWORD dwOptions;
   DWORD dwFilter;
   char achAllocModule[8];
   DWORD dwAllocBreak;
   DWORD dwAllocCount;
} WINDEBUGINFO;
```

Beschreibung

Die *WINDEBUGINFO*-Struktur enthält aktuelle System-Debug-Informationen für die Debugger-Version von Windows 3.1.

Elemente

flags: Bestimmt, welche Elemente der *WINDEBUGINFO*-Struktur gültig sind. Dieses Element kann einen oder mehrere der folgenden Werte annehmen:

Wert	Bedeutung
WDI_OPTIONS	Das Element *dwOptions* ist gültig.
WDI_FILTER	Das Element *dwFilter* ist gültig.
WDI_ALLOCBREAK	Die Elemente *achAllocModule*, *dwAllocBreak* und *dwAllocCount* sind gültig.

dwOptions: Legt Debug-Optionen fest. Dieses Element ist nur gültig, wenn *WDI_OPTIONS* im Element *flags* angegeben ist. Es kann einen oder mehrere der folgenden Werte annehmen:

Konstante	Wert/Bedeutung
DBO_CHECKHEAP	Wert 0x0001: Führt eine Überprüfung des lokalen Heap nach allen Aufrufen von Funktionen durch, die lokalen Speicher manipulieren.
DBO_BUFFERFILL	Wert 0x0004: Füllt Puffer, die an API-Funktionen übergeben werden, mit dem Wert 0xF9. Dies garantiert, daß der bereitgestellte Puffer vollständig beschreibbar ist und hilft beim Entdecken von Überschreiben-Problemen, wenn der übergebene Puffer nicht groß genug ist.
DBO_DISABLEGPTRAPPING	Wert 0x0010: Deaktiviert das Filtern fehlerhafter Interruptvektoren. Diese Option wird normalerweise nicht von Anwendungsentwicklern verwendet, weil die Gültigkeitsüberprüfung von Parametern viele fehlerverdächtige Situationen auslöst, denen kein Fehler zugrundeliegt.
DBO_CHECKFREE	Wert 0x0020: Füllt den gesamten lokalen Speicher bei seiner Freigabe mit dem Wert 0xFB. Jeder neureservierte Speicher wird überprüft, ob er noch mit 0xFB gefüllt ist. Dies garantiert, daß keine Anwendung in ein freigegebenes Speicherobjekt geschrieben hat. Diese Option zeigt keine Wirkung, wenn *DBO_CHECKHEAP* nicht angegeben wird.
DBO_INT3BREAK	Wert 0x0100: Unterbricht zum Debugger mit einem einfachem INT 3, anstatt mit einem Aufruf der Funktion *FatalExit*. Diese Option erzeugt keine Stack-Rückverfolgung.
DBO_NOFATALBREAK	Wert 0x0400: Unterbricht nicht mit dem Meldungsfenster »Abbrechen, Unterbrechen, Ignorieren«, wenn eine *DBF_FATAL*-Botschaft auftritt.

Konstante	Wert/Bedeutung
DBO_NOERRORBREAK	Wert 0x0800: Unterbricht nicht mit dem Meldungsfenster »Abbrechen, Unterbrechen, Ignorieren«, wenn eine *DBF_ERROR*-Botschaft auftritt. Diese Option bezieht sich auch auf Fehler, die durch ungültige Parameter verursacht werden.
DBO_WARNINGBREAK	Wert 0x1000: Bricht mit dem Meldungsfenster »Abbrechen, Unterbrechen, Ignorieren« ab, wenn eine *DBF_WARNING*-Botschaft auftritt. Normalerweise werden *DBF_WARNING*-Botschaften angezeigt, ohne daß eine Unterbrechung erfolgt. Diese Option bezieht sich auch auf Warnungen, die durch ungültige Parameter verursacht werden.
DBO_TRACEBREAK	Wert 0x2000: Bricht mit dem Meldungsfenster »Abbrechen, Unterbrechen, Ignorieren« bei jeder *DBF_TRACE*-Botschaft ab, die mit dem im Element *dwFilter* angegebenen Wert übereinstimmt.
DBO_SILENT	Wert 0x8000: Zeigt keine Warn- oder Fehlermeldungen an, außer in den Fällen, in denen eine Stack-Ablaufverfolgung und das Meldungsfenster »Abbrechen, Unterbrechen, Ignorieren« auftreten würde.

dwFilter: Bestimmt Filteroptionen für *DBF_TRACE*-Botschaften. Normalerweise werden Ablaufverfolgungsmeldungen nicht zum Debug-Terminal gesendet.

Dieses Element kann einen oder mehrere der folgenden Werte annehmen:

Konstante	Wert/Bedeutung
DBF_KRN_MEMMAN	Wert 0x0001: Aktiviert KERNEL-Botschaften zur lokalen und globalen Speicherverwaltung
DBF_KRN_LOADMODULE	Wert 0x0002: Aktiviert KERNEL-Botschaften zum Laden von Modulen.

Konstante	Wert/Bedeutung
DBF_KRN_SEG-MENTLOAD	Wert 0x0004: Aktiviert KERNEL-Botschaften zum Laden von Segmenten.
DBF_APPLICATION	Wert 0x0008: Aktiviert Ablaufverfolgungsbotschaften, die von einer Anwendung kommen.
DBF_DRIVER	Wert 0x0010: Aktiviert Ablaufverfolgungsbotschaften, die von einem Gerätetreiber kommen.
DBF_PENWIN	Wert 0x0020: Aktiviert Ablaufverfolgungsbotschaften, die von PENWIN kommen.
DBF_MMSYSTEM	Wert 0x0040: Aktiviert Ablaufverfolgungsbotschaften, die von MMSYSTEM kommen.
DBF_GDI	Wert 0x0400: Aktiviert Ablaufverfolgungsbotschaften, die von GDI kommen.
DBF_USER	Wert 0x0800: Aktiviert Ablaufverfolgungsbotschaften, die von USER kommen.
DBF_KERNEL	Wert 0x1000: Aktiviert jede Ablaufverfolgungsbotschaft, die vom KERNEL kommt. (Eine Kombination von *DBF_KRN_MEMMAN*, *DBF_KRN_LOADMODULE* und *DBF_KRN_SEGMENTLOAD*.)

achAllocModule: Gibt den Namen des Anwendungsmoduls an. Dieser kann vom Namen der ausführbaren Datei verschieden sein. Es darf sich nicht um den Namen einer dynamischen Linkbibliothek (DLL) handeln. Der Name ist auf acht Zeichen beschränkt.

dwAllocBreak: Bestimmt die Anzahl der lokalen und globalen Speicherreservierungen, die erlaubt werden, bevor Reservierungswünsche abgelehnt werden. Wenn die Zahl der Reservierungen den in diesem Element festgelegten Wert erreicht, schlagen diese und alle folgenden Reservierungen fehl. Wenn dieses Element Null ist, wird kein Wert für den Reservierungsabbruch gesetzt, das System zählt aber die Reservierungen und zeichnet den aktuellen Wert im Element *dwAllocCount* auf.

dwAllocCount: Aktuelle Anzahl der Reservierungen. Diese Information wird normalerweise durch einen Aufruf der Funktion *GetWinDebugInfo* abgerufen.

Hinweis

Entwickler können die Elemente *achAllocModule*, *dwAllocBreak* und *dwAllocCount* verwenden, um dafür zu sorgen, daß eine Anwendung richtig arbeitet, während nicht genügend Speicher zur Verfügung steht.

Weil vom System gemachte Speicherreservierungen fehlschlagen, sobald der Abbruchwert erreicht ist, schlagen auch Aufrufe von Funktionen wie *CreateWindow*, *CreateBrush* und *SelectObject* fehl. Nur Reservierungen, die innerhalb des vom Element *achAllocModule* angegebenen Anwendungskontexts gemacht werden, sind vom Reservierungsabbruchwert betroffen.

Siehe auch

DebugOutput, *GetWinDebugInfo*, *SetWinDebugInfo*

WINDOWPLACEMENT 3.1

Definition

```
typedef struct tagWINDOWPLACEMENT { /* wndpl */
   UINT length;
   UINT flags;
   UINT showCmd;
   POINT ptMinPosition;
   POINT ptMaxPosition;
   RECT rcNormalPosition;
} WINDOWPLACEMENT;
```

Beschreibung

Die *WINDOWPLACEMENT*-Struktur enthält Informationen über die Plazierung eines Fensters auf dem Bildschirm.

Elemente

length: Gibt die Länge der Struktur in Byte an. Die Funktion *GetWindowPlacement* gibt einen Fehlerwert zurück, wenn für dieses Element ein falscher Wert angegeben wird.

flags: Gibt Flags an, die die Position des zum Symbol verkleinerten Fensters und die Methode festlegen, durch die das Fenster wiederhergestellt wird.

Dieses Element kann einen oder beide der folgenden Werte annehmen:

Wert	Bedeutung
WPF_SETMINPOSITION	Bestimmt die x- und y-Position des zum Symbol verkleinerten Fensters. Dieses Flag muß gesetzt werden, wenn die Koordinaten in *ptMinPosition* angegeben werden.

Wert	*Bedeutung*
WPF_RESTORE-TOMAXIMIZED	Legt fest, daß das wiederhergestellte Fenster auf maximale Größe gebracht wird, unabhängig davon, ob es diese Größe hatte, bevor es verkleinert wurde. Diese Einstellung ist nur beim nächsten Wiederherstellen des Fensters gültig. Das voreingestellte Wiederherstellungsverhalten wird nicht geändert. Dieses Flag ist nur gültig, wenn der Wert *SW_SHOW-MINIMIZED* im Element *showCmd* angegeben wird.

showCmd: Bestimmt den aktuellen Anzeigestatus des Fensters. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
SW_HIDE	Verbirgt das Fenster und übergibt die Aktivierung an ein anderes Fenster.
SW_MINIMIZE	Verkleinert das angegebene Fenster auf minimale Größe und aktiviert das oberste Fenster in der Liste des Systems.
SW_RESTORE	Aktiviert ein Fenster und zeigt es an. Wenn das Fenster verkleinert oder vergrößert ist, stellt Windows es in seiner ursprünglichen Größe und an der ursprünglichen Position wieder her (entspricht *SW_SHOWNORMAL*).
SW_SHOW	Aktiviert ein Fenster und zeigt es in seiner aktuellen Größe und an seiner aktuellen Position an.
SW_SHOWMAXIMIZED	Aktiviert ein Fenster und zeigt es in maximaler Größe an.
SW_SHOWMINIMIZED	Aktiviert ein Fenster und zeigt es als Symbol an.
SW_SHOWMIN-NOACTIVE	Zeigt ein Fenster als Symbol an. Das aktuell aktive Fenster bleibt aktiv.
SW_SHOWNA	Zeigt ein Fenster in seinem aktuellen Status an. Das aktuell aktive Fenster bleibt aktiv.
SW_SHOW-NOACTIVATE	Zeigt ein Fenster in seiner letzten Größe und Position an. Das aktuell aktive Fenster bleibt aktiv.

Wert	Bedeutung
SW_SHOWNORMAL	Aktiviert und zeigt ein Fenster an. Wenn das Fenster verkleinert oder vergrößert ist, stellt es Windows in seiner ursprünglichen Größe und an seiner ursprünglichen Position wieder her (entspricht *SW_RESTORE*).

ptMinPosition: Bestimmt die Position der oberen linken Ecke des Fensters, wenn es zum Symbol verkleinert wird.

ptMaxPosition: Bestimmt die Position der oberen linken Ecke des Fensters, wenn es auf volle Größe gebracht wird.

rcNormalPosition: Bestimmt die Koordinaten des Fensters, wenn es sich in der normalen (wiederhergestellten) Position befindet.

Siehe auch *POINT*, *RECT*, *ShowWindow*

WINDOWPOS 3.1

Definition

```
typedef struct tagWINDOWPOS { /*wp */
   HWND hwnd;
   HWND hwndInsertAfter;
   int x;
   int y;
   int cx;
   int cy;
   UINT flags;
} WINDOWPOS;
```

Beschreibung Die *WINDOWPOS*-Struktur enthält Informationen über die Größe und die Position eines Fensters.

Elemente *hwnd:* Bezeichnet das Fenster.

hwndInsertAfter: Bezeichnet das Fenster, hinter dem das Fenster plaziert wird.

x: Bestimmt die x-Koordinate der linken oberen Ecke des Fensters.

y: Bestimmt die y-Koordinate der linken oberen Ecke des Fensters.

cx: Legt die Fensterbreite fest.

cy: Legt die Fensterhöhe fest.

flags: Gibt Optionen für die Fensterpositionierung an. Dieses Element kann einen der folgenden Werte annehmen:

Wert	Bedeutung
SWP_DRAWFRAME	Zeichnet einen Rahmen (definiert in der Klassenbeschreibung des Fensters) um das Fenster. Das Fenster erhält eine *WM_NCCALCSIZE*-Botschaft.
SWP_HIDEWINDOW	Verbirgt das Fenster.
SWP_NOACTIVATE	Aktiviert das Fenster nicht.
SWP_NOMOVE	Behält die aktuelle Position bei (ignoriert die Elemente *x* und *y*).
SWP_NOOWNERZ-ORDER	Ändert die Position des besitzenden Fensters in der Z-Ordnung nicht.
SWP_NOSIZE	Behält die aktuelle Größe bei (ignoriert die Elemente *cx* und *cy*).
SWP_NOREDRAW	Führt bei Änderungen kein Neuzeichnen durch.
SWP_NOREPOSITION	Entspricht *SWP_NOOWNERZORDER*.
SWP_NOZORDER	Behält die aktuelle Reihenfolge bei (ignoriert das Element *hwndInsertAfter*).
SWP_SHOWWINDOW	Zeigt das Fenster an.

Siehe auch *EndDeferWindowPos*

8. Makros

DECLARE_HANDLE 3.1

Syntax `DECLARE_HANDLE(name)`

Beschreibung Das Makro *DECLARE_HANDLE* erzeugt einen Datentyp, der zum Definieren von 16-Bit-Handles benutzt werden kann.

Parameter *name:* Bestimmt den Namen des neuen Datentyps.

Hinweis Das Makro *DECLARE_HANDLE* ist in windows.h wie folgt definiert:

```
#define DECLARE_HANDLE(name)
        struct name##__ {int unused; }; \
        typedef const struct name##__ NEAR* name
```

Siehe auch *DECLARE_HANDLE32*

DECLARE_HANDLE32 3.1

Syntax
```
#include <ddeml.h>
DECLARE_HANDLE32(name)
```

Beschreibung Das Makro *DECLARE_HANDLE32* erzeugt einen Datentyp, der zur Definition von 32-Bit-Handles benutzt werden kann.

Parameter *name:* Bestimmt den Namen des neuen Datentyps.

Hinweis Das Makro *DECLARE_HANDLE32* ist in ddeml.h wie folgt definiert:

```
#define DECLARE_HANDLE32(name)
        struct name##__ {int unused; }; \
        typedef const struct name##__ _far* name
```

Siehe auch *DECLARE_HANDLE*

FIELDOFFSET 3.1

Syntax `int FIELDOFFSET(type, field)`

Beschreibung Das Makro *FIELDOFFSET* berechnet den Offset der Adresse des angegebenen Elements in der Struktur, die durch den Parameter *type* angegeben ist.

Parameter *type:* Bestimmt den Namen der Struktur.

field: Bestimmt den Namen des Elements, das in der angegebenen Struktur definiert ist.

Rückgabewert Der Rückgabewert ist der Offset der Adresse des angegebenen Strukturelements.

Hinweis Das Makro *FIELDOFFSET* ist in windows.h wie folgt definiert:

```
#define FIELDOFFSET(type, field)  ((int)
          (&((type NEAR*)1)->field)-1)
```

GetBValue 3.1

Syntax

```
BYTE GetBValue(rgb)
DWORD rgb;            /* RGB-Farbwert                */
```

Beschreibung Das Makro *GetBValue* ermittelt den Intensitätswert des Felds für die blaue Farbe aus dem 32-Bit-Integer, der durch den Parameter *rgb* angegeben ist.

Parameter *rgb:* Bestimmt den RGB-Farbwert.

Rückgabewert Der Rückgabewert bestimmt die Intensität des Felds für die blaue Farbe.

Hinweis Das Makro *GetBValue* ist in windows.h wie folgt definiert:

```
#define GetBValue(rgb) ((BYTE)((rgb)>>16))
```

Siehe auch *GetGValue, GetRValue, RGB*

GetGValue 3.1

Syntax

```
BYTE GetGValue(rgb)
DWORD rgb;            /* RGB-Farbwert                */
```

Beschreibung Das Makro *GetGValue* ermittelt den Intensitätswert des Felds für die grüne Farbe aus dem 32-Bit-Integer, der im Parameter *rgb* angegeben ist.

Parameter *rgb:* Bestimmt den RGB-Farbwert.

Rückgabewert Der Rückgabewert bestimmt die Intensität des Felds für die grüne Farbe.

Hinweis Das Makro GetGValue ist in windows.h wie folgt definiert:

```
#define GetGValue(rgb)  ((BYTE)(((WORD)(rgb))>>8))
```

Siehe auch *GetBValue, GetRValue, RGB*

GetRValue 3.1

Syntax
```
BYTE GetRValue(rgb)
DWORD rgb;          /* RGB-Farbwert          */
```

Beschreibung Das Makro *GetRValue* ermittelt den Intensitätswert des Felds für die rote Farbe aus dem 32-Bit-Integer, der im Parameter *rgb* angegeben ist.

Parameter *rgb:* Bestimmt den RGB-Farbwert.

Rückgabewert Der Rückgabewert bestimmt die Intensität des Felds für die rote Farbe.

Hinweis Das Makro *GetRValue* ist in windows.h wie folgt definiert:

```
#define GetRValue(rgb)  ((BYTE)(rgb))
```

Siehe auch *GetBValue, GetGValue, RGB*

MAKELP 3.1

Syntax
```
void FAR* MAKELP(wSet, wOff)
WORD wSel;          /* Selektor              */
WORD wOff;          /* Offset                */
```

Beschreibung Das Makro *MAKELP* kombiniert einen Segmentselektor und den Offset einer Adresse, um einen **long**-Zeiger (32 Bit) auf eine Speicheradresse zu erzeugen.

Parameter *wSel:* Bestimmt einen Segment-Selektor.

wOff: Bestimmt einen Offset, ausgehend vom Anfang des angegebenen Segments bis zum gewünschten Byte.

Rückgabewert Der Rückgabewert ist ein **long**-Zeiger auf einen typenlosen Datentyp.

Hinweis Das Makro *MAKELP* ist in windows.h wie folgt definiert:

```
#define MAKELP(sel, off) ((void FAR*)
          MAKELONG((off), (sel)))
```

Siehe auch *MAKELONG*

MAKELPARAM 3.1

Syntax
```
LPARAM MAKELPARAM(wLow, wHigh)
WORD wLow;          /* niederwertiges WORD   */
WORD wHigh;         /* höherwertiges WORD    */
```

Beschreibung Das Makro *MAKELPARAM* erzeugt einen vorzeichenlosen **long**-Integer, der als Parameter *lParam* in einer Botschaft verwendet werden kann. Dieses Makro verknüpft zwei Integerwerte, die in den Parametern *wLow* und *wHigh* angegeben sind.

Parameter *wLow:* Bestimmt das niederwertige *WORD* des neuen **long**-Werts.

wHigh: Bestimmt das höherwertige *WORD* des neuen **long**-Werts.

Rückgabewert Der Rückgabewert bestimmt einen vorzeichenlosen **long**-Integerwert.

Hinweis Das Makro *MAKELPARAM* ist in windows.h wie folgt definiert:

```
#define MAKELPARAM(low, high) ((LPARAM)
          MAKELONG(low, high))
```

Siehe auch *MAKELONG, MAKELRESULT*

MAKELRESULT 3.1

Syntax

```
LRESULT MAKELRESULT(wLow, wHigh)

WORD wLow;          /* niederwertiges WORD          */
WORD wHigh;         /* höherwertiges WORD           */
```

Beschreibung Das Makro *MAKELRESULT* erzeugt einen vorzeichenlosen **long**-Integer, der als Rückgabewert einer Fensterfunktion verwendet werden kann. Das Makro verknüpft zwei Integerwerte, die in den Parametern *wLow* und *wHigh* angegeben sind.

Parameter *wLow:* Bestimmt das niederwertige *WORD* des neuen **long**-Werts.

wHigh: Bestimmt das höherwertige *WORD* des neuen **long**-Werts.

Rückgabewert Der Rückgabewert bestimmt einen vorzeichenlosen **long**-Integerwert.

Hinweis Das Makro *MAKELRESULT* ist in windows.h wie folgt definiert:

```
#define MAKELRESULT(low, high) ((LRESULT)
          MAKELONG(low, high))
```

Siehe auch *MAKELONG, MAKELPARAM*

OFFSETOF 3.1

Syntax

```
WORD OFFSETOF(lp)

void FAR* lp;       /* long-Zeiger                  */
```

Beschreibung Das Makro *OFFSETOF* ruft den Adressoffset des angegebenen **long**-Zeigers ab.

Parameter *lp:* Bestimmt einen **long**-Zeiger.

Rückgabewert Der Rückgabewert ist der Offset-Anteil der Adresse.

Hinweis Das Makro *OFFSETOF* ist in windows.h wie folgt definiert:

```
#define OFFSETOF(lp)   LOWORD(lp)
```

Siehe auch *LOWORD, SELECTOROF*

SELECTOROF 3.1

Syntax

```
WORD SELECTOROF(lp)
void FAR* lp;          /* long-Zeiger */
```

Beschreibung Das Makro *SELECTOROF* ruft den Segmentselektor des angegebenen **long**-Zeigers ab.

Parameter *lp:* Bestimmt einen **long**-Zeiger.

Rückgabewert Der Rückgabewert ist der Segmentselektor.

Hinweis Das Makro *SELECTOROF* ist in windows.h wie folgt definiert:

```
#define SELECTOROF(lp)   HIWORD(lp)
```

Siehe auch *HIWORD, OFFSETOF*

9. Escape-Codes

MOUSETRAILS

Syntax

```
short Escape(hdc, MOUSETRAILS, sizeof(WORD),
  lpTrailSize, NULL)

HDC hdc;
LPINT lpTrailSize;
```

Beschreibung Der Escape-Code *MOUSETRAILS* aktiviert oder deaktiviert die Mausspur für Anzeigegeräte.

Parameter *hdc:* Bezeichnet den Gerätekontext.

lpTrailSize: Zeigt auf eine 16-Bit-Variable, deren Inhalt die Aktion und die Anzahl der Mauszeigerabbilder (Spurgröße) bestimmt.

Die Variable kann einen der im folgenden beschriebenen Werte enthalten:

Wert	Bedeutung
1 bis 7	Aktiviert die Mausspur und setzt die Spurgröße auf die angegebene Zahl. Ein Wert von 1 fordert einen einzelnen Mauszeiger an. Der Wert 2 fordert einen zusätzlichen Mauszeiger an, der hinter dem aktuellen Mauszeiger dargestellt werden soll. Die größtmögliche Zahl für Mauszeigerbilder ist 7. Der Escape-Code setzt den Eintrag *MouseTrails* in der Datei WIN.INI auf den angegebenen Wert und gibt die neue Größe der Mausspur zurück.
0	Deaktiviert die Mausspur. Der Escape-Code setzt den Eintrag *MouseTrails* auf den negativen Wert der aktuellen Spurgröße – wenn diese positiv ist – und gibt den negativen Wert zurück.
-1	Aktiviert die Mausspur. Der Gerätetreiber liest den Eintrag *MouseTrails* im Abschnitt [windows] der Datei WIN.INI. Wenn der Wert des Eintrags positiv ist, setzt der Escape-Code die Spurgröße auf den angegebenen Wert.

Wert	Bedeutung
	Ist der Eintrag negativ, setzt der Escape-Code die Spurgröße auf den absoluten Wert des Eintrags und schreibt den positiven Wert in die Datei WIN.INI. Wenn der Eintrag *MouseTrails* nicht gefunden wird, setzt der Escape-Code die Spurgröße auf 7 und schreibt einen neuen Eintrag *MouseTrails* mit dem Wert 7 in die Datei WIN.INI. Der Escape-Code gibt dann die neue Spurgröße zurück.
-2	Deaktiviert die Mausspur, veranlaßt den Gerätetreiber aber nicht zur Aktualisierung der Datei WIN.INI.
-3	Aktiviert die Mausspur, veranlaßt den Gerätetreiber aber nicht zur Aktualisierung der Datei WIN.INI.

Rückgabewert Wenn die Funktion erfolgreich ist, bestimmt der Rückgabewert die neue Spurgröße. Der Rückgabewert ist Null, wenn der Escape-Code nicht unterstützt wird.

POSTSCRIPT_DATA

Beschreibung Der Drucker-Escape-Code *POSTSCRIPT_DATA* ist mit dem Escape-Code *PASSTROUGH* identisch.

POSTSCRIPT_IGNORE

Syntax
```
short Escape(hdc, POSTSCRIPT_IGNORE, NULL, lpfOutput,
   NULL)

HDC hdc;
BOOL FAR* lpfOutput;
```

Beschreibung Der Drucker-Escape-Code *POSTSCRIPT_IGNORE* setzt ein Flag, das festlegt, ob Ausgaben unterdrückt werden.

Parameter *hdc:* Bezeichnet den Gerätekontext.

lpfOutput: Zeigt auf ein Flag, das anzeigt, ob die Ausgabe unterdrückt werden soll. Dieser Wert ist ungleich Null, wenn die Ausgabe unterdrückt werden soll, andernfalls Null.

Rückgabewert Der Rückgabewert ist die vorherige Einstellung des Ausgabe-Flag.

Hinweis Anwendungen, die eigenen PostScript-Code erzeugen, können den Escape-Code *POSTSCRIPT_IGNORE* verwenden, um die Erzeugung von Ausgaben durch den PostScript-Gerätetreiber zu verhindern.

SETALLJUSTVALUES

Syntax

```
short Escape(hdc, SETALLJUSTVALUES,
   sizeof(EXTTEXTDATA), lpInData, NULL)

HDC hdc;
EXTTEXTDATA FAR* lplnData;
```

Beschreibung

Die Benutzung des Drucker-Escape-Code *SETALLJUSTVALUES* wird nicht empfohlen. Anwendungen sollten stattdessen die Funktion *ExtTextOut* verwenden. Dieser Escape-Code setzt alle Textausrichtungswerte, die für Textausgaben unter Windows 3.0 und früher verwendet werden.

Textausrichtung meint hier die Einfügung zusätzlicher Pixel zwischen Trennzeichen innerhalb einer Textzeile. Normalerweise wird das Leerzeichen als Trennzeichen benutzt.

Parameter

hdc: Bezeichnet den Gerätekontext.

lpInData: Zeigt auf eine *EXTTEXTDATA*-Struktur, die die Textausrichtungswerte definiert. Weitere Informationen über diese Struktur finden Sie im »Hinweis«.

Rückgabwert

Wenn der Escape-Code erfolgreich ist, ist der Rückgabewert 1, andernfalls Null.

Hinweis

Der Parameter *lplnData* zeigt auf eine *EXTTEXTDATA*-Struktur, die die Textausrichtungswerte beschreibt, die für Textausgaben verwendet werden. Die *EXTTEXTDATA*-Struktur hat folgende Form:

```
typedef struct{
    short nSize;
    LPALLJUSTREC lpInData;
    LPFONTINFO lpFont;
    LPTEXTXFORM lpXForm;
    LPDRAWMODE lpDrawMode;
} EXTTEXTDATA;
```

Diese Struktur enthält eine *JUST_VALUE_STRUCT*-Struktur mit der folgenden Form:

```
typedef struct {
    short nCharExtra;
    WORD cch;
    short nBreakExtra;
    WORD nBreakCount;
}JUST_VALUE_STRUCT;
```

Im folgenden finden Sie eine Beschreibung der Elemente der *JUST_VALUE_STRUCT*-Struktur:

nCharExtra: Bestimmt die Gesamtgröße in Geräteeinheiten, die auf *cch* Zeichen verteilt werden muß.

cch: Bestimmt die Anzahl Zeichen, auf die der in *nCharExtra* angegebene Platz verteilt werden muß.

nBreakExtra: Bestimmt den gesamten Platz in Geräteeinheiten, der auf *nBreakCount* Zeichen verteilt werden soll.

nBreakCount: Bestimmt die Anzahl von Trennzeichen, auf die der im Element *nBreakExtra* angegebene Platz verteilt werden soll.

Die Einheiten für die Elemente *nCharExtra* und *nBreakExtra* sind die Schrifteinheiten des Geräts und hängen davon ab, ob relative Zeichenbreiten mit dem Escape-Code *ENABLE-RELATIVEWIDTHS* aktiviert wurden.

Die mit diesem Escape-Code gesetzten Werte wirken sich auf nachfolgende Aufrufe der Funktion *TextOut* aus. Der Treiber unterbricht die Verteilung des zusätzlichen Platzes, der im Element *nCharExtra* angegeben ist, wenn die im Element *nCharCount* angegebene Anzahl Zeichen ausgegeben wurde. Entsprechend beendet er die Verteilung des Platzes, der im Element *nBreakExtra* angegeben ist, wenn die Anzahl Zeichen ausgegeben wurde, die im Element *nBreakCount* angegeben ist. Wird die Funktion *GetTextExtent* unmittelbar nach dem Aufruf der Funktion *TextOut* für den gleichen String aufgerufen, findet die Bearbeitung in der gleichen Weise statt.

Um die Ausrichtung mit den Funktionen *SetTextJustification* und *SetTextCharacterExtra* zu reaktivieren, sollten Anwendungen den Escape-Code *SETALLJUSTVALUES* aufrufen und die Elemente *nCharExtra* und *nBreakExtra* auf Null setzen.

10. DDE-Transaktionen

Die Bibliothek DDEML (Dynamic Data Exchange Management Library) benachrichtigt Anwendungen über einen dynamischen Datenaustausch (DDE), der diese Anwendung betrifft, indem sie Transaktionen an die DDE-Callback-Funktion der Anwendung sendet. Eine Transaktion ähnelt einer Botschaft. Es handelt sich um eine benannte Konstante, die in ihren Parametern zusätzliche Informationen über die Transaktion enthält. Dieses Kapitel enthält die DDE-Transaktionen in alphabethischer Reihenfolge.

XTYP_ADVDATA 3.1

Syntax

```
#include <ddeml.h>
XTYP_ADVDATA

hszTopic = hsz1;       /* Handle des String mit dem
                          Themennamen                */
hszItem = hsz2;        /* Handle des String mit dem
                          Elementnamen               */
hDataAdvise = hData;   /* Handle der Advise-Daten    */
```

Beschreibung Die DDE-Callback-Funktion eines Client kann diese Transaktion empfangen, nachdem der Client eine Advise-Schleife mit einem Server eingerichtet hat. Diese Transaktion informiert den Client darüber, daß der Wert des Datenelements verändert wurde.

Parameter *hszTopic:* Wert von *hsz1*. Bezeichnet den Themennamen.

hszItem: Wert von *hsz2*. Bezeichnet den Elementnamen.

hDataAdvise: Wert von *hData*. Bezeichnet die Daten, die mit dem angegebenen Thema und Element verknüpft sind. Wenn der Client das Flag *XTYPF_NODATA* bei der Einrichtung der Advise-Schleife angegeben hat, ist dieser Parameter *NULL*.

Rückgabewert Eine DDE-Callback-Funktion sollte *DDE_FACK* zurückgeben, wenn sie diese Transaktion bearbeitet. Sie sollte *DDE_FBUSY* zurückgeben, wenn sie zu beschäftigt ist, um die Transaktion zu bearbeiten oder *DDE_FNOTPROCESSED*, wenn sie die Transaktion verweigert.

Hinweis Anwendungen müssen das Daten-Handle nicht freigeben, das sie während dieser Transaktion erhalten haben. Wenn die Anwendung die Daten nach der Rückkehr der Callback-Funktion bearbeiten muß, muß sie die Daten kopieren. Hierzu können Anwendungen die Funktion *DdeGetData* verwenden.

Siehe auch *DdeClientTransaction, DdePostAdvise*

XTYP_ADVREQ 3.1

Syntax

```
#include <ddeml.h>
XTYP_ADVREQ

hszTopic = hsz1;            /* Handle des String für
                               den Themennamen      */
hszItem = hsz2;             /* Handle des String für
                               den Elementnamen     */
cAdvReq = LOWORD(dwData1);  /* Zähler für die verblei-
                               benden Transaktionen */
```

Beschreibung Das System sendet diese Transaktion an einen Server, nachdem der Server die Funktion *DdePostAdvise* aufgerufen hat. Diese Transaktion teilt dem Server mit, daß eine Transaktion für das angegebene Thema und den angegebenen Namen vorliegt, und daß die zugehörigen Daten geändert wurden.

Parameter *hszTopic:* Wert von *hsz1*. Bezeichnet den Themennamen.

hszItem: Wert von *hsz2*. Bezeichnet den Elementnamen, dessen zugehörige Daten geändert wurden.

cAdvReq: Wert des niederwertigen *WORD* von *dwData1*. Bestimmt den Zähler für die *XTYP_ADVREQ*-Transaktionen, die für dieselbe Kombination aus Themenname, Elementname und Formatname noch bearbeitet werden müssen. Dieser Zähler bezieht sich auf den aktuellen Aufruf der Funktion *DdePostAdvise*. Wenn die aktuelle *XTYP_ADVREQ*-Transaktion die letzte ist, ist der Zähler Null. Server können mit Hilfe dieses Zählers ermitteln, ob sie ein *HDATA_APPOWNED*-Daten-Handle für die Advise-Daten erzeugen sollen.

Hat die DDEML die Transaktion *XTYP_ADVREQ* wegen eines zu spät angekommenen Transaktions-Flag *DDE_FACK* des Client gesendet, ist das niederwertige *WORD* auf *CADV_LATEACK* gesetzt. Das Transaktions-Flag *DDE_FACK* kommt verspätet an, wenn ein Server Informationen schneller sendet, als der Client sie bearbeiten kann.

Rückgabewert Der Server sollte die Funktion *DdeCreateDataHandle* aufrufen, um ein Daten-Handle zu erzeugen und zurückzugeben, das die veränderten Daten bezeichnet. Wenn der Server die Transaktion nicht vollenden kann, sollte er *NULL* zurückgeben.

Hinweis Server können diesen Transaktionstyp nicht sperren. Der Rückgabewert *CBR_BLOCK* wird ignoriert.

Siehe auch *DdeCreateDataHandle, DdeInitialize, DdePostAdvise*

XTYP_ADVSTART 3.1

Syntax
```
#include <ddeml.h>
XTYP_ADVSTART

hszTopic = hsz1;      /* Handle des String für den
                         Themennamen                */
hszItem = hsz2;       /* Handle des String für den
                         Elementnamen               */
```

Beschreibung Die DDE-Callback-Funktion eines Server erhält diese Transaktion, wenn ein Client *XTYP_ADVSTART* für den Parameter *wType* der Funktion *DdeClientTransaction* angibt. Ein Client benutzt diese Transaktion, um eine Advise-Schleife mit einem Server einzurichten.

Parameter *hszTopic:* Wert von *hsz1*. Bezeichnet den Themennamen.

hszItem: Wert von *hsz2*. Bezeichnet den Elementnamen.

Rückgabewert Um eine Advise-Schleife für das angegebene Thema und das angegebene Element zuzulassen, sollte die DDE-Callback-Funktion eines Server einen Wert ungleich Null zurückgeben. Um die Advise-Schleife zu verhindern, sollte sie Null zurückgeben. Wenn die Callback-Funktion einen Wert ungleich Null zurückgibt, lösen nachfolgende Aufrufe der Funktion *DdePostAdvice* durch den Server das Senden einer Transaktion *XTYP_ADVREQ* vom System an den Server aus.

Hinweis Wenn ein Client eine Advise-Schleife für eine bestimmte Kombination aus Themenname, Elementname und Formatname anfordert, für die bereits eine Advise-Schleife eingerichtet wurde, erzeugt die DDEML kein Duplikat dieser Schleife. Stattdessen ändert die DDEML die Flags *XTYPF_ACKREQ* und *XTYPF_NODATA* der Advise-Schleife, damit sie der letzten Anforderung entsprechen.

Hat die Server-Anwendung das Flag *CBF_FAIL_ADVISES* in der Funktion *DdeInitialize* angegeben, wird diese Transaktion gefiltert.

Siehe auch *DdeClientTransaction, DdeInitialize, DdePostAdvise*

XTYP_ADVSTOP 3.1

Syntax
```
#include <ddeml.h>
XTYP_ADVSTOP
```

```
hszTopic = hsz1;      /* Handle des String für den
                         Themennamen                */
hszItem = hsz2;       /* Handle des String für den
                         Elementnamen               */
```

Beschreibung Die DDE-Callback-Funktion eines Server empfängt diese Transaktion, wenn ein Client *XTYP_ADVSTOP* im Parameter *wType* der Funktion *DdeClientTransaction* angibt. Ein Client benutzt diese Transaktion, um eine Advise-Schleife mit dem Server zu beenden.

Parameter *hszTopic:* Wert von *hsz1*. Bezeichnet den Themennamen.

hszItem: Wert von *hsz2*. Bezeichnet den Elementnamen.

Rückgabewert Diese Transaktion hat keinen Rückgabewert.

Hinweis Wenn die Server-Anwendung das Flag *CBF_FAIL_ADVISES* in der Funktion *DdeInitialize* angegeben hat, wird diese Transaktion gefiltert.

Siehe auch *DdeClientTransaction*, *DdeInitialize*, *DdePostAdvise*

XTYP_CONNECT 3.1

Syntax

```
#include <ddeml.h>
XTYP_CONNECT
hszTopic = hsz1;                 /* Handle des String
                                    für den
                                    Themennamen     */
hszService = hsz2;               /* Handle des String
                                    für den
                                    Servicenamen    */
pcc = (CONVCONTEXT FAR*) dwData1; /* Adresse der
                                    CONVCONTEXT-
                                    Struktur        */
fSameInst = (BOOL) dwData2;      /* Instanzen-
                                    identitäts-Flag */
```

Beschreibung Die DDE-Callback-Funktion eines Server empfängt diese Transaktion, wenn ein Client einen Servivenamen angibt, den der Server unterstützt. Der Themenname darf im Aufruf der Funktion *DdeConnect* nicht auf *NULL* gesetzt sein.

Parameter *hszTopic:* Wert von *hsz1*. Bezeichnet den Themennamen.

hszService: Wert von *hsz2*. Bezeichnet den Servicenamen.

pcc: Wert von *dwData1*. Zeigt auf eine *CONVCONTEXT*-Datenstruktur, die Kontextinformationen für die Konversation enthält. Wenn der Client keine DDEML-Anwendung ist, sollte dieser Parameter auf Null gesetzt werden.

fSameInst: Wert von *dwData2*. Bestimmt, ob der Client die gleiche Anwendungsinstanz wie der Server ist. Dies ist der Fall, wenn der Parameter den Wert *TRUE* hat. Ist der Parameter *FALSE*, handelt es sich um eine andere Instanz.

Rückgabewert Damit der Client eine Konversation für den angegebenen Service- und Themennamen einrichten kann, sollte die DDE-Callback-Funktion des Server einen Wert ungleich Null zurückgeben. Um die Konversation zu verhindern, sollte sie Null zurückgeben. Wenn die Callback-Funktion einen Wert ungleich Null zurückgibt und die Konversation erfolgreich eingerichtet wird, übergibt das System das Handle der Konversation an den Server, indem sie die Transaktion *XTYP_CONNECT_CONFIRM* an die DDE-Callback-Funktion des Server sendet. Letzteres geschieht nicht, wenn der Server das Flag *CBF_FAIL_CONNECT_CONFIRMS* in der Funktion *DdeInitialize* angegeben hat.

Hinweis Wenn die Server-Anwendung das Flag *CBF_FAIL_CONNECTIONS* in der Funktion *DdeInitialize* angegeben hat, wird die Transaktion gefiltert.

Ein Server kann diesen Transaktionstyp nicht sperren. Der Rückgabewert *CBR_BLOCK* wird ignoriert.

Siehe auch *DdeConnect*, *DdeInitialize*

XTYP_CONNECT_CONFIRM 3.1

Syntax

```
#include <ddeml.h>
XTYP_CONNECT_CONFIRM

hszTopic = hsz1;              /* Handle des String für
                                 den Themennamen      */
hszService = hsz2;            /* Handle des String für
                                 den Servicenamen     */
fSameInst = (BOOL) dwData2;   /* Instanzenidentitäts-
                                 Flag                 */
```

Beschreibung Die DDE-Callback-Funktion eines Server empfängt diese Transaktion als Bestätigung der Einrichtung einer Konversation mit einem Client. Außerdem erhält der Server so das Handle der Konversation. Das System sendet diese Transaktion aufgrund einer vorhergehenden Transaktion *XTYP_CONNECT* oder *XTYP_WILDCONNECT*.

Parameter *hszTopic:* Wert von *hsz1*. Bezeichnet den Themennamen, für den die Konversation eingerichtet wurde.

hszService: Wert von *hsz2*. Bezeichnet den Servicenamen, für den die Konversation eingerichtet wurde.

fSameInst: Wert von *dwData2*. Bestimmt, ob der Client die gleiche Anwendungsinstanz wie der Server ist. Wenn dieser Parameter einen Wert ungleich Null hat, ist der Client die gleiche Instanz. Ist der Parameter Null, handelt es sich beim Client um eine andere Instanz.

Rückgabewert Diese Transaktion hat keinen Rückgabewert.

Hinweis Wenn die Server-Anwendung das Flag *CBF_FAIL_CONFIRMS* in der Funktion *DdeInitialize* angegeben hat, wird diese Transaktion gefiltert.

Ein Server kann diesen Transaktionstyp nicht sperren. Der Rückgabewert *CBR_BLOCK* wird ignoriert.

Siehe auch *DdeConnect*, *DdeConnectList*, *DdeInitialize*

XTYP_DISCONNECT 3.1

Syntax

```
#include <ddeml.h>
XTYP_DISCONNECT

fSameInst = (BOOL) dwData2;   /* Instanzenidentitäts-
                                 Flag                */
```

Beschreibung Die DDE-Callback-Funktion einer Anwendung empfängt diese Transaktion, wenn der Partner der Anwendung in einer Konversation die Funktion *DdeDisconnect* aufruft, um die Konversation zu beenden.

Parameter *fSameInst:* Wert von *dwData2*. Bestimmt, ob die Partner in der Konversation die gleichen Anwendungsinstanzen sind. Wenn dieser Parameter *TRUE* ist, handelt es sich um identische Instanzen. Ist der Parameter *FALSE*, sind die Partner unterschiedliche Instanzen.

Rückgabewert Diese Transaktion hat keinen Rückgabewert.

Hinweis Wenn die Anwendung das Flag *CBF_SKIP_DISCONNECTS* in der Funktion *DdeInitialize* angegeben hat, wird diese Transaktion gefiltert.

Die Anwendung kann den Status der beendeten Konversation ermitteln, indem sie die Funktion *DdeQueryConvInfo* aufruft, während diese Transaktion bearbeitet wird. Das Handle der Konversation wird ungültig, sobald die Callback-Funktion zurückkehrt.

Anwendungen können diesen Transaktionstyp nicht sperren. Der Rückgabewert *CBR_BLOCK* wird ignoriert.

Siehe auch *DdeDisconnect*, *DdeQueryConvInfo*

XTYP_ERROR 3.1

Syntax

```
#include <ddeml.h>
XTYP_ERROR

wErr = LOWORD(dwData1);   /* Fehlerwert              */
```

Beschreibung Eine DDE-Callback-Funktion empfängt diese Transaktion, wenn ein kritischer Fehler auftritt.

Parameter *wErr:* Wert von *dwData1*. Bestimmt den Fehlerwert. Aktuell wird nur der Fehlerwert *DMLERR_LOW_MEMORY* unterstützt. Dieser Fehlerwert bedeutet, daß der Speicher nicht ausreicht. Die Folge kann der Verlust von Advise-, Poke- oder Execute-Daten oder ein Systemabsturz sein.

Rückgabewert Diese Transaktion hat keinen Rückgabewert.

Hinweis Anwendungen können diesen Transaktionstyp nicht sperren. Der Rückgabewert *CBR_BLOCK* wird ignoriert. Die DDEML versucht Speicher freizugeben, indem sie unkritische Ressourcen entfernt. Anwendungen mit gesperrten Konversationen sollten versuchen, die Sperre aufzuheben.

XTYP_EXECUTE 3.1

Syntax

```
#include <ddeml.h>
XTYP_EXECUTE

hszTopic = hsz1;     /* Handle des String für den
                        Themennamen                */
hDataCmd = hData;    /* Handle des Befehls-String  */
```

Beschreibung Die DDE-Callback-Funktion eines Server empfängt diese Transaktion, wenn ein Client *XTYP_EXECUTE* für den Parameter *wType* der Funktion *DdeClientTransaction* angibt. Ein Client verwendet diese Transaktion, um einen Befehls-String an den Server zu senden.

Parameter *hszTopic:* Wert von *hsz1*. Bezeichnet den Themennamen.

hDataCmd: Wert von *hData*. Bezeichnet den Befehls-String.

Rückgabewert Die DDE-Callback-Funktion eines Server sollte *DDE_FACK* zurückgeben, wenn sie diese Transaktion bearbeitet. Wenn sie die Transaktion nicht bearbeitet, weil sie beschäftigt ist, sollte sie *DDE_FBUSY* zurückgeben. Wenn sie die Transaktion aus anderen Gründen nicht akzeptiert, sollte sie *DDE_FNOTPROCESSED* zurückgeben.

Hinweis Wenn die Server-Anwendung das Flag *CBF_FAIL_EXECUTES* in der Funktion *DdeInitialize* angegeben hat, wird diese Transaktion gefiltert.

Anwendungen müssen das Daten-Handle nicht freigeben, das sie während dieser Transaktion erhalten haben. Wenn Anwendungen den String bearbeiten müssen, nachdem die Callback-Funktion zurückgekehrt ist, müssen sie den Befehls-String kopieren. Anwendungen können die Funktion *DdeGetData* benutzen, um die Daten zu kopieren.

Siehe auch *DdeClientTransaction, DdeInitialize*

XTYP_MONITOR 3.1

Syntax

```
#include <ddeml.h>
XTYP_MONITOR

hDataEvent = hData;    /* Handle der Ereignisdaten    */
fwEvent = dwData2;     /* Ereignis-Flag               */
```

Beschreibung Die DDE-Callback-Funktion einer DDE-überwachenden Anwendung empfängt diese Transaktion, sobald ein DDE-Ereignis im System auftritt. Anwendungen können diese Transaktion nur empfangen, wenn sie das Flag *APPCLASS_MONITOR* beim Aufruf der Funktion *DdeInitialize* angegeben haben.

Parameter *hDataEvent:* Wert von *hData*. Bezeichnet ein globales Speicherobjekt, das Informationen über das DDE-Ereignis enthält. Anwendungen sollten die Funktion *DdeAccessData* benutzen, um einen Zeiger auf das Objekt zu erhalten.

fwEvent: Wert von *dwData2*. Bestimmt das DDE-Ereignis. Dieser Parameter kann einen der im folgenden beschriebenen Werte annehmen:

Wert	Bedeutung
MF_CALLBACKS	Das System hat eine Transaktion an eine DDE-Callback-Funktion gesendet. Das globale Speicherobjekt enthält eine *MONCBSTRUCT*-Struktur, die Informationen über die Transaktion bereitstellt.
MF_CONV	Eine DDE-Konversation wurde eingerichtet oder beendet. Das globale Speicherobjekt enthält eine *MONCONVSTRUCT*-Struktur, die Informationen über die Konversation bereitstellt.
MF_ERRORS	Ein DDE-Fehler ist aufgetreten. Das globale Speicherobjekt enthält eine *MONERRSTRUCT*-Struktur, die Informationen über den Fehler bereitstellt.

Wert	Bedeutung
MF_HSZ_INFO	Eine DDE-Anwendung hat ein String-Handle erzeugt oder freigegeben oder den Verwendungszähler eines String-Handle erhöht. Möglich ist auch, daß das String-Handle als Ergebnis eines Aufrufs der Funktion *DdeUnInitialize* freigegeben und entfernt wurde. Das globale Speicherobjekt enthält eine *MONHSZSTRUCT*-Struktur, die Informationen über das String-Handle bereitstellt.
MF_LINKS	Eine DDE-Anwendung hat eine Advise-Schleife gestartet oder beendet. Das globale Speicherobjekt enthält eine *MONLINKSTRUCT*-Struktur, die Informationen über die Advise-Schleife bereitstellt.
MF_POSTMSGS	Das System oder eine Anwendung hat eine DDE-Botschaft übergeben. Das globale Speicherobjekt enthält eine *MONMSGSTRUCT*-Struktur, die Informationen über die Botschaft bereitstellt.
MF_SENDMSGS	Das System oder eine Anwendung hat eine DDE-Botschaft gesendet. Das globale Speicherobjekt enthält eine *MONMSGSTRUCT*-Struktur, die Informationen über die Botschaft bereitstellt.

Rückgabewert Die Callback-Funktion sollte Null zurückgeben, wenn sie diese Transaktion bearbeitet.

Siehe auch *DdeAccessData*, *DdeInitialize*

XTYP_POKE 3.1

Syntax

```
#include <ddeml.h>
XTYP_POKE

hszTopic = hsz1;      /* Handle des String für den
                         Themennamen                  */
hszItem = hsz2;       /* Handle des String für den
                         Elementnamen                 */
hDataPoke = hData;    /* Handle der Daten für den
                         Server                       */
```

Beschreibung Die DDE-Callback-Funktion eines Server empfängt diese Transaktion, wenn ein Client *XTYP_POKE* als Parameter *wType* der Funktion *DdeClientTransaction* angibt. Ein Client verwen-

det diese Transaktion, um nicht angeforderte Daten an den Server zu übergeben.

Parameter *hszTopic:* Wert von *hsz1*. Bezeichnet den Themennamen.

hszItem: Wert von *hsz2*. Bezeichnet den Elementnamen.

hDataPoke: Wert von *hData*. Bezeichnet die Daten, die der Client an den Server sendet.

Rückgabewert Die DDE-Callback-Funktion eines Server sollte *DDE_FACK* zurückgeben, wenn sie diese Transaktion bearbeitet, *DDE_FBUSY*, wenn sie zu beschäftigt ist, um diese Transaktion zu bearbeiten oder *DDE_FNOTPROCESSED*, wenn sie die Bearbeitung dieser Transaktion verweigert.

Hinweis Wenn die Server-Anwendung das Flag *CBF_FAIL_POKES* in der Funktion *DdeInitialize* angegeben hat, wird diese Transaktion gefiltert.

Siehe auch *DdeClientTransaction*, *DdeInitialize*

XTYP_REGISTER 3.1

Syntax

```
#include <ddeml.h>
XTYP_REGISTER

hszBaseServName = hsz1;   /* Handle des String für
                             den eigentlichen
                             Servicenamen          */
hszInstServName = hsz2;   /* Handle des String für
                             den Servicenamen
                             der Instanz           */
```

Beschreibung Eine DDE-Callback-Funktion empfängt diesen Transaktionstyp, sobald ein DDEML-Server die Funktion *DdeNameService* benutzt, um einen Servicenamen zu registrieren. Dies geschieht auch, wenn eine Nicht-DDEML-Anwendung gestartet wird, die das Thema *System* unterstützt.

Parameter *hszBaseServName:* Wert von *hsz1*. Bezeichnet den zu registrierenden eigentlichen Servicenamen.

hszInstServName: Wert von *hsz2*. Bezeichnet den instanzenspezifischen Servicenamen, der registriert wird.

Rückgabewert Diese Transaktion hat keinen Rückgabewert.

Hinweis Wenn die Anwendung das Flag *CBF_SKIP_REGISTRATIONS* in der Funktion *DdeInitialize* angegeben hat, wird diese Transaktion gefiltert.

Anwendungen können diesen Transaktionstyp nicht sperren. Der Rückgabewert *CBR_BLOCK* wird ignoriert.

Anwendungen sollten den Parameter *hszBaseServName* benutzen, um den Servicenamen der Liste der für den Benutzer verfügbaren Server hinzuzufügen. Sie sollten den Parameter *hszInstServName* benutzen, um die gestartete Anwendungsinstanz zu bezeichnen.

Siehe auch *DdeInitialize, DdeNameService*

XTYP_REQUEST 3.1

Syntax

```
#include ddeml.h>
XTYP_REQUEST

hszTopic = hsz1;     /* Handle des String für den
                        Themennamen                 */
hszItem = hsz2;      /* Handle des String für den
                        Elementnamen                */
```

Beschreibung Die DDE-Callback-Funktion eines Server empfängt diese Transaktion, wenn ein Client *XTYP_REQUEST* für den Parameter *wType* der Funktion *DdeClientTransaction* angibt. Ein Client verwendet diese Transaktion, um Daten vom Server anzufordern.

Parameter *hszTopic:* Wert von *hsz1*. Bezeichnet den Themennamen.

hszItem: Wert von *hsz2*. Bezeichnet den Elementnamen, der geändert wurde.

Rückgabewert Der Server sollte die Funktion *DdeCreateDataHandle* aufrufen, um ein Daten-Handle zu erzeugen, das die geänderten Daten bezeichnet. Die Funktion sollte dieses Handle zurückgeben. Der Server sollte *NULL* zurückgeben, wenn er die Transaktion nicht vollenden kann. Gibt der Server *NULL* zurück, empfängt der Client das Bestätigungs-Flag *DDE_FNOTPROCESSED*.

Hinweis Wenn die Server-Anwendung das Flag *CBF_FAIL_REQUESTS* in der Funktion *DdeInitialize* angegeben hat, wird diese Transaktion gefiltert.

Nimmt das Beantworten dieser Transaktion viel Zeit in Anspruch, kann der Server *CBR_BLOCK* zurückgeben, um weitere Transaktionen in der aktuellen Konversation zu unterbinden. Er kann die Transaktion dann asynchron bearbeiten. Wenn der Server die Bearbeitung beendet hat und die Daten an den Client übergeben werden können, kann der Server die Funktion *DdeEnableCallback* aufrufen, um die Konversation wiederaufzunehmen.

Siehe auch *DdeClientTransaction, DdeCreateDataHandle, DdeEnableCallback, DdeInitialize*

XTYP_UNREGISTER 3.1

Syntax

```
#include <ddeml.h>
XTYP_UNREGISTER

hszBaseServName = hsz1;    /* Handle des String
                              für den eigentlichen
                              Servicenamen          */
hszInstServName = hsz2;    /* Handle des String
                              für den Servicenamen
                              der Instanz           */
```

Beschreibung Eine DDE-Callback-Funktion empfängt diese Transaktion, wenn ein DDEML-Server die Funktion *DdeNameService* benutzt, um die Registrierung eines Servicenamens aufzuheben. Das gleiche geschieht, wenn eine Nicht-DDEML-Anwendung beendet wird, die das Thema *System* unterstützt.

Parameter *hszBaseServName:* Wert von *hsz1*. Bezeichnet den eigentlichen Servicenamen, dessen Registrierung aufgehoben werden soll.

hszInstServName: Wert von *hsz2*. Bezeichnet den instanzenspezifischen Servicenamen, dessen Registrierung aufgehoben werden soll.

Rückgabewert Diese Transaktion hat keinen Rückgabewert.

Hinweis Wenn die Anwendung das Flag *CBF_SKIP_REGISTRATIONS* in der Funktion *DdeInitialize* angegeben hat, wird diese Transaktion gefiltert.

Anwendungen können diesen Transaktionstyp nicht sperren. Der Rückgabewert *CBR_BLOCK* wird ignoriert.

Anwendungen sollten den Parameter *hszBaseServName* benutzen, um den Servicenamen aus der Liste der für den Benutzer verfügbaren Server zu entfernen. Sie sollten den Parameter *hszInstServName* benutzen, um die Anwendungsinstanz zu bezeichnen, die beendet wurde.

Siehe auch *DdeInitialize*, *DdeNameService*

XTYP_WILDCONNECT 3.1

Syntax

```
#include <ddeml.h>
XTYP_WILDCONNECT

hszTopic = hsz1;                /* Handle des
                                   String für den
                                   Themennamen    */
hszService = hsz2;              /* Handle des
                                   String für den
                                   Servicenamen   */
```

```
pcc = (CONVCONTEXT FAR*)dwData1;    /* Adresse der
                                       CONVCONTEXT-
                                       Struktur      */
fSameInst = (BOOL) dwData2;         /* Instanzeniden-
                                       titäts-Flag   */
```

Beschreibung Die DDE-Callback-Funktion eines Server empfängt diese Transaktion, wenn ein Client einen Servicenamen mit dem Wert *NULL*, einen Themennamen mit dem Wert *NULL* oder beides in einem Aufruf der Funktion *DdeConnect* angibt. Diese Transaktion ermöglicht es dem Client, eine Konversation mit dem Server über jedes Paar aus Service- und Themenname einzurichten, die den angegebenen Servicenamen und Themennamen entspricht.

Parameter *hszTopic:* Wert von *hsz1*. Bezeichnet den Themennamen. Wenn dieser Parameter *NULL* ist, fordert der Client eine Konversation über alle Themen an, die der Server unterstützt.

hszService: Wert von *hsz2*. Bezeichnet den Servicenamen. Wenn dieser Parameter *NULL* ist, fordert der Client eine Konversation über alle Servicenamen an, die der Server unterstützt.

pcc: Wert von *dwData1*. Zeigt auf eine *CONVCONTEXT*-Datenstruktur, die Kontextinformationen über die Konversation enthält. Wenn der Client keine DDEML-Anwendung ist, wird dieser Parameter auf Null gesetzt.

fSameInst: Wert von *dwData2*. Bestimmt, ob der Client die gleiche Anwendungsinstanz wie der Server ist. Wenn dieser Parameter *TRUE* ist, ist der Client die gleiche Instanz. Ist der Parameter *FALSE*, ist der Client eine andere Instanz.

Rückgabewert Der Server sollte ein Daten-Handle zurückgeben, das ein Array aus *HSZPAIR*-Strukturen bezeichnet. Dieses Array sollte eine Struktur für jedes Service-/Themenname-Paar enthalten, die dem durch den Client angeforderten Paar entsprechen. Das Array muß mit einem *NULL*-String-Handle beendet werden. Das System sendet die Transaktion *XTYP_CONNECT_CONFIRM* an den Server, um jede Konversation zu bestätigen und um die Konversations-Handles an den Server zu übergeben. Wenn der Server das Flag *CBF_SKIP_CONNECT_CONFIRMS* in der Funktion *DdeInitialize* angegeben hat, kann er diese Bestätigungen nicht empfangen.

Um die Transaktion *XTYP_WILDCONNECT* zurückzuweisen, sollte der Server *NULL* zurückgeben.

Hinweis Wenn die Serveranwendung in der Funktion *DdeInitialize* das Flag *CBF_FAIL_CONNECTIONS* angegeben hat, dann wird diese Transaktion gefiltert.

Server können diesen Transaktionstyp nicht sperren. Der Rückgabewert *CBR_BLOCK* wird ignoriert.

Siehe auch *DdeConnect*, *DdeInitialize*

XTYP_XACT_COMPLETE 3.1

Syntax

```
#include <ddeml.h>
XTYP_XACT_COMPLETE

hszTopic = hsz1;       /* Handle des String für den
                          Themennamen                */
hszItem = hsz2;        /* Handle des String für den
                          Elementnamen               */
dDataXactID = hData;   /* Handle der
                          Transaktionsdaten          */
dwXactID = dwData1;    /* Transaktionsbezeichner     */
fwStatus = dwData2;    /* Status-Flag                */
```

Beschreibung Die DDE-Callback-Funktion eines Client empfängt diese Transaktion, wenn eine asynchrone Transaktion beendet wird, die durch die Funktion *DdeClientTransaction* eingeleitet wurde.

Parameter *hszTopic:* Wert von *hsz1*. Bezeichnet den Themennamen, auf den sich die abgeschlossene Transaktion bezogen hat.

hszItem: Wert von *hsz2*. Bezeichnet den Elementnamen, auf den sich die abgeschlossene Transaktion bezogen hat.

hDataXact: Wert von *hData*. Bezeichnet die Daten, auf die sich die abgeschlossene Transaktion bezogen hat, wenn diese verfügbar sind. Dieser Parameter ist *TRUE*, wenn die Transaktion erfolgreich war, obwohl keine Daten einbezogen waren. Ist die Transaktion fehlgeschlagen, hat dieser Parameter den Wert *NULL*.

dwXactID: Wert von *dwData1*. Enthält den Transaktionsbezeichner der abgeschlossenen Transaktion.

fwStatus: Wert von *dwData2*. Enthält alle verfügbaren *DDE_*-Status-Flags im niederwertigen *WORD*. Das unterstützt Anwendungen, die vom Bit *DDE_APPSTATUS* abhängig sind. Anwendungen sollten diese Bits nicht weiter verwenden. Zukünftige Versionen der DDEML werden sie voraussichtlich nicht unterstützen.

Rückgabewert Diese Funktion hat keinen Rückgabewert.

Hinweis Anwendungen müssen das Daten-Handle nicht freigeben, das sie während der Transaktion erhalten haben. Sollen Anwendungen die Daten bearbeiten, nachdem die Callback-Funktion zurückgekehrt ist, müssen sie die Daten kopieren. Anwendungen können die Daten mit Hilfe der Funktion *DdeGetData* kopieren.

Siehe auch *DdeClientTansaction*

11. Ereignisse und Botschaften des Datei-Managers

Der Datei-Manager kommuniziert mit einer Erweiterungs-DLL für den Datei-Manager, indem er Ereignisse und Menübefehle an die Funktion *FMExtensionProc* der DLL sendet. Während der Bearbeitung eines Ereignisses oder eines Befehls kann die DLL Informationen vom Datei-Manager abrufen, indem sie Datei-Manager-Botschaften mit Hilfe der Funktion *SendMessage* sendet. Dieses Kapitel enthält eine komplette Beschreibung der Ereignisse und Botschaften für den Datei-Manager von Microsoft Windows 3.1.

Datei-Manager-Ereignisse

Dieser Abschnitt enthält die Ereignisse des Datei-Managers in alphabetischer Reihenfolge.

FMEVENT_INITMENUE

Beschreibung	Die Botschaft *FMEVENT_INITMENUE* wird an eine Erweiterungs-DLL gesendet, wenn der Benutzer das Menü für die Erweiterung aus der Menüleiste des Datei-Managers auswählt. Die Erweiterung kann diese Benachrichtigung benutzen, um die Menüelemente ihres eigenen Menüs zu initialisieren.
Parameter	*lParam:* Bestimmt das Menü-Handle im höherwertigen *WORD*. Das niederwertige *WORD* bestimmt den Deltawert für das Menüelement.
Rückgabewert	Diese Botschaft hat keinen Rückgabewert.
Hinweis	Eine Erweiterung empfängt diese Botschaft nur, wenn der Benutzer das Hauptmenü auswählt. Enthält die Erweiterung Untermenüs, muß sie diese zur gleichen Zeit wie das Hauptmenü initialisieren.
Siehe auch	*FMExtensionProc*

FMEVENT_LOAD

Beschreibung Die Botschaft *FMEVENT_LOAD* wird an eine Erweiterungs-DLL gesendet, wenn der Datei-Manager die DLL lädt.

Parameter *lParam:* Zeigt auf eine *FMS_LOAD*-Struktur, die den Deltawert des Menüelements bestimmt. Eine Erweiterungs-DLL sollte den Deltawert des Menüelements speichern und die anderen Strukturelemente mit Informationen über die Erweiterung füllen. Die *FMS_LOAD*-Struktur hat folgende Form:

```
typdef struct tagFMS_LOAD { /* fmsld */
   DWORD dwSize;
   char szMenuName[MENU_TEXT_LEN];
   HMENU hMenu;
   UINT wMenuDelta;
} FMS_LOAD;
```

Rückgabewert Diese Botschaft hat keinen Rückgabewert.

Hinweis Eine Anwendung sollte die Elemente *dwSize*, *szMenuName* und *hMenu* füllen. Sie sollte außerdem den Wert des Elements *wMenuDelta* speichern und ihn zum Bezeichnen von Menüelementen benutzen, wenn das Menü verändert wird. Weitere Informationen finden Sie in der Beschreibung der *FMS_LOAD*-Struktur in Kapitel 7 »Strukturen«.

Siehe auch *FMSExtensionProc*

FMEVENT_SELCHANGE

Beschreibung Die Botschaft *FMEVENT_SELCHANGE* wird an eine Erweiterungs-DLL gesendet, wenn der Anwender einen Dateinamen im Verzeichnisfenster des Datei-Managers oder im Fenster *Suchergebnis* auswählt.

Parameter *lParam:* Wird nicht verwendet.

Rückgabewert Diese Botschaft hat keinen Rückgabewert.

Hinweis Veränderungen in der Darstellung der Baumstruktur des Verzeichnisfensters erzeugen diese Botschaft nicht.

Da der Benutzer die Auswahl mehrmals ändern kann, muß die Erweiterungs-DLL unmittelbar nach der Bearbeitung dieser Botschaft zurückkehren, damit die Auswahl durch den Benutzer nicht verlangsamt wird.

Siehe auch *FMExtensionProc*

FMEVENT_UNLOAD

Beschreibung	Die Botschaft *FMEVENT_UNLOAD* wird an eine Erweiterungs-DLL gesendet, wenn der Datei-Manager die DLL aus dem Speicher entfernt.
Parameter	*lParam:* Wird nicht verwendet.
Rückgabewert	Diese Botschaft hat keinen Rückgabewert.
Hinweis	Die Werte *hwnd* und *hMenu*, die mit den Botschaften *FMEVENT_LOAD* und *FMEVENT_INITMENU* übergeben werden, sind zum Zeitpunkt dieser Botschaft möglicherweise bereits ungültig.
Siehe auch	*EMExtensionProc*

FMEVENT_USER_REFRESH

Beschreibung	Die Botschaft *FMEVENT_USER_REFRESH* wird an die Erweiterungs-DLL gesendet, wenn der Benutzer den Befehl *Aktualisieren* im Menü *Fenster* des Datei-Managers auswählt. Die Erweiterung kann diese Benachrichtigung benutzen, um ihr Menü zu aktualisieren.
Parameter	*lParam:* Wird nicht verwendet.
Rückgabewert	Diese Botschaft hat keinen Rückgabewert.
Siehe auch	*FMExtensionProc*

Datei-Manager-Botschaften

Dieser Abschnitt enthält die Botschaften des Datei-Managers in alphabetischer Reihenfolge.

FM_GETDRIVEINFO

Beschreibung	Eine Erweiterung des Datei-Managers sendet eine *FM_GETDRIVEINFO*-Botschaft, um Laufwerksinformationen aus dem aktiven Fenster des Datei-Managers abzurufen.
Parameter	*wParam:* Wird nicht verwendet. *lParam:* Zeigt auf eine *FMS_GETDRIVEINFO*-Struktur, die die Laufwerksinformationen aufnimmt. Die *FMS_GETDRIVEINFO*-Struktur hat folgende Form:

```
typdef struct tagFMS_GETDRIVEINFO {/* fmsgdi */
   DWORD dwTotalSpace;
   DWORD dwFreeSpace;
   char szPath[260];
   char szVolume[14];
   char szShare[128];
} FMS_GETDRIVEINFO, FAR* LPFMS_GETDRIVEINFO;
```

Rückgabewert Der Rückgabewert ist immer ungleich Null.

Siehe auch *FMExtensionProc, FM_GETDRIVEINFO*

FM_GETFILESEL

Beschreibung Eine Erweiterung des Datei-Managers sendet eine *FM_GETFILESEL*-Botschaft, um Informationen über die ausgewählte Datei im aktiven Fenster des Datei-Managers abzurufen. Dabei kann es sich um das Verzeichnisfenster oder das Fenster *Suchergebnis* handeln.

Parameter *wParam:* Bestimmt den von Null ausgehenden Index der ausgewählten Datei, über die Informationen abgerufen werden sollen.

lParam: Zeigt auf eine *FMS_FILESEL*-Struktur, die Informationen über die Auswahl aufnimmt. Die *FMS_FILESEL*-Struktur hat folgende Form:

```
typdef struct tagFMS_GETFILESEL {/* fmsgfs */
   UINT wTime;
   UINT wDate;
   DWORD dwSize;
   BYTE bAttr;
   char szName[260];
} FMS_GETFILESEL;
```

Rückgabewert Der Rückgabewert ist der von Null ausgehende Index der ausgewählten Datei, über die Informationen abgerufen wurden.

Hinweis Eine Erweiterung kann die Botschaft *FM_GETSELCOUNT* benutzen, um den Zähler der ausgewählten Dateien zu erhalten.

Das Element *szName* der *FMS_GETFILESEL*-Struktur besteht aus einem OEM-Zeichen-String. Bevor dieser String dargestellt wird, sollten Erweiterungen die Funktion *OemToAnsi* benutzen, um den String in einen Windows-ANSI-Zeichen-String zu konvertieren. Soll der String an das Dateisystem (MS-DOS) übergeben werden, sollte die Erweiterung den String nicht konvertieren.

Siehe auch *FMExtensionProc, FM_GETFILESELLFN, FM_GETSELCOUNT, FM_GETSELCOUNTLFN, OemToAnsi*

FM_GETFILESELLFN

Beschreibung Eine Erweiterung des Datei-Managers sendet eine *FM_GETFILESELLFN*-Botschaft, um Informationen über eine ausgewählte Datei im aktiven Fenster des Datei-Managers abzurufen. Dabei kann es sich um das Verzeichnisfenster oder das Fenster *Suchergebnis* handeln. Die ausgewählte Datei kann einen vollständigen Netzdateinamen besitzen.

Parameter *wParam:* Bestimmt den von Null ausgehenden Index der ausgewählten Datei, über die Informationen abgerufen werden sollen.

lParam: Zeigt auf eine *FMS_GETFILESEL*-Struktur, die Informationen über die Datei empfängt. Die *FMS_GETFILESEL*-Struktur hat folgende Form:

```
typdef struct tagFMS_GETFILESEL {/* fmsgfs */
   UINT wTime;
   UINT wDate;
   DWORD dwSize;
   BYTE bAttr;
   char szName[260];
} FMS_GETFILESEL;
```

Rückgabewert Der Rückgabewert ist der von Null ausgehende Index der ausgewählten Datei, über die Informationen abgerufen wurden.

Hinweis Diese Botschaft sollte nur von Erweiterungen benutzt werden, die vollständige Netzdateinamen unterstützen. Dies könnte zum Beispiel für Netzerweiterungen gelten.

Erweiterungen können die Botschaft *FM_GETSELCOUNT* benutzen, um die Anzahl der ausgewählten Dateien zu erhalten.

Das Element *szName* der *FMS_GETFILESEL*-Struktur enthält einen OEM-Zeichen-String. Bevor dieser String dargestellt wird, sollten Erweiterungen die Funktion *OemToAnsi* aufrufen, um den String in einen Windows-ANSI-Zeichen-String zu konvertieren. Soll der String an das Dateisystem (MS-DOS) weitergegeben werden, sollte die Erweiterung den String nicht konvertieren.

Siehe auch *FMExtension*, *FM_GETFILESEL*, *FM_GETSELCOUNT*, *FM_GETSELCOUNTLFN*, *OemToAnsi*

FM_GETFOCUS

Bescheibung Eine Erweiterung des Datei-Managers sendet eine *FM_GETFOCUS*-Botschaft, um den Typ des Fensters des Datei-Managers abzurufen, das den Fokus besitzt.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Der Rückgabewert bestimmt den Typ des Fensters des Datei-Managers, das den Fokus hat. Möglich sind folgende Werte:

Wert	Bedeutung
FM_FOCUS_DIR	Der Verzeichnisteil eines Verzeichnisfensters.
FM_FOCUS_TREE	Der Teil eines Verzeichnisfensters, der die Baumstruktur enthält.
FM_FOCUS_DRIVES	Die Laufwerksleiste eines Verzeichnisfensters.
FM_FOCUS_SEARCH	Das Fenster *Suchergebnis*.

FM_GETSELCOUNT

Beschreibung Eine Erweiterung des Datei-Managers sendet eine *FM_GETSELCOUNT*-Botschaft, um die Anzahl der ausgewählten Dateien im Verzeichnisfenster oder im Fenster *Suchergebnis* abzurufen. Letzteres hängt davon ab, welches das aktive Fenster ist.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Der Rückgabewert ist die Anzahl der ausgewählten Dateien.

Siehe auch *FM_GETFILESEL*, *FM_GETFILESELLFN*, *FM_GETSELCOUNTLFN*

FM_GETSELCOUNTLFN

Beschreibung Eine Erweiterung des Datei-Managers sendet eine *FM_GETSELCOUNTLFN*-Botschaft, um die Anzahl der ausgewählten Dateien im Verzeichnisfenster oder im Fenster *Suchergebnis* abzurufen. Letzteres hängt davon ab, welches das aktive Fenster ist. Der Zähler schließt Dateien ein, die vollständige Netzdateinamen besitzen.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Der Rückgabewert ist die Anzahl der ausgewählten Dateien.

Hinweis Nur Erweiterungen, die vollständige Netzdateinamen unterstützen, sollten diese Botschaft verwenden. Dies könnte zum Beispiel für Netzerweiterungen gelten.

Siehe auch *FM_GETFILESEL*, *FM_GETFILESELLFN*, *FM_GETSELCOUNT*

FM_REFRESH_WINDOWS

Beschreibung Eine Erweiterung des Datei-Managers sendet eine *FM_REFRESH_WINDOWS*-Botschaft, um den Datei-Manager zu veranlassen, entweder sein aktives Fenster oder alle Fenster neu zu zeichnen.

Parameter *wParam:* Bestimmt, ob der Datei-Manager sein aktives Fenster oder alle Fenster neu zeichnet. Wenn dieser Parameter ungleich Null ist, zeichnet der Datei-Manager alle Fenster neu. Ist dieser Parameter Null, zeichnet der Datei-Manager nur das aktive Fenster neu.

lParam: Wird nicht verwendet.

Rückgabewert Der Rückgabewert dieser Botschaft hat keine Bedeutung.

Hinweis Führt eine Erweiterung Änderungen am Dateisystem durch, wird dies vom Datei-Manager automatisch bemerkt. Erweiterungen sollten diese Botschaft nur in Situationen benutzen, in denen Laufwerksverbindungen erstellt oder abgebrochen werden.

Siehe auch *FMExtensionProc*

FM_RELOAD_EXTENSIONS

Beschreibung Eine Erweiterung des Datei-Managers – oder eine andere Anwendung – sendet eine *FM_RELOAD_EXTENSIONS*-Botschaft, um den Datei-Manager zum Neuladen aller Erweiterungs-DLLs zu veranlassen, die im Abschnitt [AddOns] in der Datei WINFILE.INI aufgeführt sind.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Der Rückgabewert dieser Botschaft hat keine Bedeutung.

Hinweis Andere Anwendungen können die Funktion *PostMessage* benutzen, um diese Botschaft an den Datei-Manager zu senden. Um das Handle des entsprechenden Fensters des Datei-Managers zu erhalten, können diese Anwendungen *WFS_FRAME* als Parameter *lpszClassName* in einem Aufruf der Funktion *FindWindow* übergeben.

Siehe auch *FindWindow*, *FMExtensionProc*, *PostMessage*

12. Botschaften der Systemsteuerung

Die Systemsteuerung kommuniziert durch Botschaften, die sie an die Einsprungfunktion *CPlApplet* sendet, mit einer dynamischen Linkbibliothek (DLL) der Systemsteuerung. Eine Botschaft besteht aus drei Teilen: einer Botschaftsnummer und zwei 32-Bit-Parametern. Die Botschaftsnummern werden durch vordefinierte Botschaftsnamen bezeichnet. Die beiden 32-Bit-Parameter enthalten botschaftsabhängige Werte.

Dieses Kapitel enthält eine alphabetische Liste aller Botschaften, die von der Einsprungfunktion *CPlApplet* empfangen werden können. Um diese Botschaften verwenden zu können, muß die Header-Datei cpl.h aufgenommen werden.

CPL_DBLCLK 3.1

Beschreibung	Die Botschaft *CPL_DBLCLK* wird an eine DLL der Systemsteuerung gesendet, wenn der Anwender einen Doppelklick auf dem Symbol einer Anwendung ausführt, die von der DLL unterstützt wird.
Parameter	*lParam1:* Bestimmt die Nummer der Anwendung. Diese Nummer muß zwischen Null und dem Wert liegen, der um eins kleiner ist als die Antwort, die auf die Botschaft *CPL_GETCOUNT* zurückgeliefert wurde.
	lParam2: Bestimmt den Wert, der für die Anwendung in das Element *lData* geladen wird.
Rückgabewert	Die DLL der Systemsteuerung liefert Null zurück, wenn sie diese Botschaft erfolgreich bearbeitet hat.
Hinweis	Als Antwort auf diese Botschaft sollte die DLL der Systemsteuerung das Dialogfenster der Anwendung anzeigen.
Siehe auch	*CPL_GETCOUNT*

CPL_EXIT 3.1

Beschreibung	Die Botschaft *CPL_EXIT* wird einmal an eine DLL der Systemsteuerung gesendet, bevor die Systemsteuerung die Funktion *FreeLibrary* aufruft, um die DLL freizugeben.

Parameter *lParam1:* Wird nicht verwendet.

lParam2: Wird nicht verwendet.

Rückgabewert Die DLL der Systemsteuerung liefert Null zurück, wenn sie diese Botschaft erfolgreich bearbeitet hat.

Hinweis Als Antwort auf diese Botschaft sollte eine DLL der Systemsteuerung den gesamten Speicher freigeben, den sie reserviert hat, und die nötige Neuorganisation des Speichers durchführen.

CPL_GETCOUNT 3.1

Beschreibung Die Botschaft *CPL_GETCOUNT* ermittelt die Anzahl der Anwendungen, die eine DLL der Systemsteuerung unterstützt.

lParam1: Wird nicht verwendet.

lParam2: Wird nicht verwendet.

Rückgabewert Die DLL der Systemsteuerung liefert die Anzahl der Anwendungen zurück, die sie unterstützt.

Hinweis Diese Botschaft wird unmittelbar im Anschluß an die Botschaft *CPL_INIT* gesendet.

Siehe auch *CPL_INIT*

CPL_INIT 3.1

Beschreibung Die Botschaft *CPL_INIT* fordert eine DLL der Systemsteuerung auf, eine globale Initialisierung – insbesondere Speicherreservierung – durchzuführen.

Parameter *lParam1:* Wird nicht verwendet.

lParam2: Wird nicht verwendet.

Rückgabewert Die DLL der Systemsteuerung liefert einen Wert ungleich Null zurück, wenn die Initialisierung erfolgreich war. Andernfalls liefert sie Null zurück. Wenn die DLL Null zurückliefert, ruft die Systemsteuerung die Funktion *FreeLibrary* auf und beendet die Kommunikation mit der DLL.

Hinweis Die DLL sollte als Antwort auf diese Botschaft Speicher reservieren. Nur so kann sie eine Fehlerbedingung anzeigen.

Diese Botschaft wird unmittelbar nach dem Laden der DLL gesendet.

CPL_INQUIRE

Beschreibung Die Botschaft *CPL_INQUIRE* wird an eine DLL der Systemsteuerung gesendet, um Informationen über eine Anwendung anzufordern, die von der DLL unterstützt wird.

Diese Botschaft wird für die Abwärtskompatibilität von Microsoft Windows Version 3.1 bereitgestellt. Eine Anwendung sollte die Botschaft *CPL_NEWINQUIRE* anstelle der Botschaft *CPL_INQUIRE* verwenden.

Parameter *lParam1:* Bestimmt die Nummer der Anwendung. Diese Nummer muß zwischen Null und dem Wert liegen, der um eins kleiner ist als die Antwort, die die Botschaft *CPL_GETCOUNT* zurückgeliefert.

lParam2: Bestimmt einen **far**-Zeiger auf eine *CPLINFO*-Struktur. Die DLL sollte diese Struktur mit den Ressourcenbezeichnern für das Symbol, den Kurznamen, der Beschreibung und allen anwendungsdefinierten Werten initialisieren, die der Anwendung zugeordnet sind. Die *CPLINFO*-Struktur hat folgende Form:

```
#include <cpl.h>

typedef struct tagCPLINFO { /*  cpli  */
   int idIcon;
   int idName;
   int idInfo;
   LONG lData;
} CPLINFO;
```

Rückgabewert Die DLL der Systemsteuerung liefert Null zurück, wenn sie diese Botschaft erfolgreich bearbeitet hat.

Hinweis Diese Botschaft wird an jede Anwendung einmal gesendet, die von der DLL unterstützt wird. Sie wird unmittelbar nach der Botschaft *CPL_GETCOUNT* gesendet. Eine DLL kann eine Initialisierung auf Anwendungsebene vornehmen, wenn sie diese Botschaft erhält. Als Reaktion auf die Botschaft *CPL_INIT* sollte Speicher reserviert werden.

Siehe auch *CPL_GETCOUNT*, *CPL_INIT*, *CPL_NEWINQUIRE*

CPL_NEWINQUIRE

Beschreibung Die Botschaft *CPL_NEWINQUIRE* wird an eine DLL der Systemsteuerung gesendet, um Informationen über eine Anwendung abzurufen, die von DLL unterstützt wird.

Parameter *lParam1:* Bestimmt die Nummer der Anwendung. Diese Nummer muß zwischen Null und dem Wert liegen, der um eins

kleiner ist als die Antwort, die auf die Botschaft *CPL_GETCOUNT* zurückgeliefert wurde.

lParam2: Bestimmt einen **far**-Zeiger auf eine *NEWCPLINFO*-Struktur. Die DLL sollte diese Struktur mit den Informationen über die Anwendung initialisieren. Die *NEWCPLINFO*-Struktur hat folgende Form:

```
#include <cpl.h>

typedef struct tagCPLINFO { /*  ncpli  */
   DWORD dwSize;
   DWORD dwFlags;
   DWORD dwHelpContext;
   LONG lData;
   HICON hIcon;
   char szName[32];
   char szInfo[64];
   char szHelpFile[128];
} NEWCPLINFO;
```

Rückgabewert Die DLL der Systemsteuerung liefert Null zurück, wenn sie diese Botschaft erfolgreich bearbeitet hat.

Hinweis Diese Botschaft wird einmal an jede Anwendung gesendet, die von der DLL unterstützt wird. Sie wird unmittelbar nach der Botschaft *CPL_GETCOUNT* gesendet. Eine DLL kann beim Empfang in einer **switch**-Anweisung Initialisierungen auf Anwendungsebene durchführen. Als Antwort auf die Botschaft *CPL_INIT* sollte Speicher reserviert werden.

Siehe auch *CPL_GETCOUNT*, *CPL_INIT*, *CPL_INQUIRE*

CPL_SELECT

Beschreibung Die Botschaft *CPL_SELECT* wird an eine DLL der Systemsteuerung gesendet, wenn der Anwender das Symbol einer Anwendung auswählt, die von der DLL unterstützt wird.

Parameter *lParam1:* Bestimmt die Nummer der Anwendung.

lParam2: Bestimmt den anwendungsdefinierten Wert, der für die Anwendung in das Element *lData* geladen wird.

Rückgabewert Die DLL der Systemsteuerung liefert Null zurück, wenn sie diese Botschaft erfolgreich bearbeitet hat.

CPL_STOP

Beschreibung Die Botschaft *CPL_STOP* wird an jede Anwendung einmal gesendet, wenn die Systemsteuerung beendet wird.

Parameter *lParam1:* Bestimmt die Nummer der Anwendung. Diese Nummer muß zwischen Null und dem Wert liegen, der um eins kleiner ist als die Antwort, die auf die Botschaft *CPL_GETCOUNT* zurückgeliefert wurde.

lParam2: Bestimmt den anwendungsdefinierten Wert, der für die Anwendung in das Element *lData* geladen wird.

Rückgabewert Die DLL der Systemsteuerung liefert Null zurück, wenn sie diese Botschaft erfolgreich bearbeitet hat.

Hinweis Als Antwort auf diese Botschaft sollte eine DLL der Systemsteuerung eine anwendungsspezifische Neuorganisation des Speichers durchführen.

Siehe auch *CPL_GETCOUNT*

WM_CPL_LAUNCH

Beschreibung Eine Anwendung sendet die Botschaft *WM_CPL_LAUNCH* an die Systemsteuerung, um den Start einer Anwendung der Systemsteuerung anzufordern.

Parameter *wParam:* Bestimmt das Handle das Fensters, das diese Botschaft sendet. Die Botschaft *WM_CPL_LAUNCHED* wird an dieses Fenster gesendet.

lParam: Bestimmt einen **far**-Zeiger auf einen String, der den Namen der zu öffnenden Anwendung enthält.

Rückgabewert Der Rückgabewert ist ungleich Null, wenn die Anwendung gestartet wurde. Andernfalls ist er Null.

Hinweis Der String, auf den der Parameter *lParam* zeigt, muß in einem globalen Speicherobjekt enthalten sein, das mit dem Flag *GMEM_NOT_BANKED* reserviert wurde.

Siehe auch *WM_CPL_LAUNCHED*

WM_CPL_LAUNCHED

Beschreibung Die Botschaft *WM_CPL_LAUNCHED* wird gesendet, wenn eine Anwendung der Systemsteuerung beendet wurde, die durch die *WM_CPL_LAUNCH*-Botschaft gestartet wurde. Die Botschaft *WM_CPL_LAUNCHED* wird an das Fenster gesendet, das durch den Parameter *wParam* der Botschaft *WM_CPL_LAUNCH* bezeichnet wurde.

Parameter *wParam:* Gibt an, ob die Anwendung gestartet wurde. Wenn die Anwendung gestartet wurde, hat dieser Parameter einen Wert ungleich Null. Andernfalls ist er Null.

lParam: Wird nicht verwendet.

Rückgabewert Der von der Anwendung zurückgelieferte Wert wird für diese Botschaft ignoriert.

Siehe auch *WM_CPL_LAUNCH*

13. Botschaften der Standarddialogfenster

Ein Standarddialogfenster sendet eine Botschaft, um Anwendungen darüber zu informieren, daß der Anwender eine Auswahl im Dialogfenster vorgenommen oder geändert hat. Diese Botschaften können von Anwendungen dazu benutzt werden, anwendungsdefinierte Aktionen auszuführen. Beispiele hierfür sind das Zurückweisen von Benutzereingaben oder das Setzen anwenderdefinierter Farben.

Bevor eine Anwendung Standarddialogfensterbotschaften benutzen kann, muß sie diese Botschaften und die in diesem Kapitel angegebenen Botschaftskonstanten registrieren, die in der Header-Datei commdlg.h definiert sind. Dies erfolgt durch Aufruf der Funktion *RegisterWindowMessage*.

In diesem Kapitel sind die Botschaften der Standarddialogfenster beschrieben. Sie sind in alphabetischer Reihenfolge aufgeführt.

COLOROKSTRING 3.1

Beschreibung	Die Botschaft *COLOROKSTRING* wird unmittelbar vor dem Schließen des Dialogfensters *Farben* an die Filterfunktion der Anwendung gesendet. Indem man der Anwendung die Möglichkeit gibt, das Dialogfenster *Farben* geöffnet zu lassen, obwohl der Anwender bereits den *OK*-Schalter betätigt hat, wird eine bessere Kontrolle der anwenderderdefinierte Farben ermöglicht.
Parameter	*wParam:* Wird nicht verwendet. *lParam:* Zeigt auf eine *CHOOSECOLOR*-Struktur, die die aktuell gewählte Farbe enthält.
Rückgabewert	Wenn die Anwendung einen Wert ungleich Null zurückliefert, nachdem sie diese Botschaft bearbeitet hat, wurde das Dialogfenster nicht geschlossen.
Hinweis	Um die Botschaft *COLOROKSTRING* verwenden zu können, muß die Anwendung einen neuen Botschaftsbezeichner erzeugen. Dies erfolgt durch den Aufruf der Funktion *RegisterWindowMessage*, wobei die Konstante *COLOROKSTRING* als einziger Parameter übergeben wird.

Siehe auch *RegisterWindowMessage*

FILEOKSTRING 3.1

Beschreibung Die Botschaft *FILEOKSTRING* wird vom Dialogfenster *Datei öffnen* oder *Datei speichern unter* an die Filterfunktion der Anwendung gesendet, wenn der Anwender einen Dateinamen ausgewählt und den *OK*-Schalter betätigt hat. Die Botschaft ermöglicht es der Anwendung, den vom Anwender gewählten Dateinamen anzunehmen oder zurückzuweisen.

Parameter *wParam:* Wird nicht verwendet.

lParam: Zeigt auf eine *OPENFILENAME*-Struktur, die die Informationen über die Auswahl des Anwenders enthält. Diese Information schließt den ausgewählten Dateinamen ein.

Rückgabewert Die Filterfunktion sollte den Wert -1 zurückliefern, wenn sie den vom Anwender gewählten Dateinamen zurückgewiesen hat. In diesem Fall bleibt das Dialogfenster offen und der Anwender muß einen neuen Dateinamen auswählen. Sie sollte 0 zurückliefern, wenn sie den vom Anwender gewählten Dateinamen angenommen hat oder diese Botschaft nicht bearbeitet.

Hinweis Um die Botschaft *FILEOKSTRING* verwenden zu können, muß die Anwendung einen neuen Botschaftsbezeichner erzeugen. Dies erfolgt durch Aufruf der Funktion *RegisterWindowMessage*, wobei die Konstante *FILEOKSTRING* als einziger Parameter übergeben wird.

Siehe auch *RegisterWindowMessage*

FINDMSGSTRING 3.1

Beschreibung Die Botschaft *FINDMSGSTRING* wird vom Dialogfenster *Suchen* oder *Ersetzen* an die Anwendung gesendet, wenn der Anwender einen String eingegeben und den *OK*-Schalter gedrückt hat. Diese Botschaft enthält die vom Anwender in den Steuerelementen des Dialogfensters angegebenen Daten, wie z.B. die Richtung, in der die Anwendung nach einem String suchen soll, ob die Anwendung die Groß- oder Kleinschreibung des Strings beachten soll oder ob die Anwendung den String als vollständiges Wort suchen soll.

Parameter *wParam:* Wird nicht verwendet.

lParam: Zeigt auf eine *FINDREPLACE*-Struktur, die die Informationen über die Eingaben des Anwenders enthält.

Rückgabewert Die Anwendung sollte Null zurückliefern.

Hinweis Um die Botschaft *FINDMSGSTRING* verwenden zu können, muß die Anwendung einen neuen Botschaftsbezeichner erzeugen. Dies erfolgt durch Aufruf der Funktion *RegisterWindowMessage*, wobei die Konstante *FINDMSGSTRING* als einziger Parameter übergeben wird.

Siehe auch *RegisterWindowMessage*

HELPMSGSTRING 3.1

Beschreibung Die Botschaft *HELPMSGSTRING* wird vom Standarddialogfenster an die Fensterfunktion des besitzenden Fensters gesendet, wenn der Anwender den *Hilfe*-Schalter gedrückt hat. So kann eine Anwendung eine anwenderdefinierte Hilfe für das Dialogfenster bereitstellen.

Parameter *wParam:* Wird nicht verwendet.

lParam: Zeigt auf eine Struktur, die das Standarddialogfenster beschreibt.

Rückgabewert Die Anwendung liefert Null zurück.

Hinweis Um die Botschaft *HELPMSGSTRING* verwenden zu können, muß die Anwendung einen neuen Botschaftsbezeichner erzeugen. Dies erfolgt durch einen Aufruf der Funktion *RegisterWindowMessage*, wobei die Konstante *HELPMSGSTRING* als einziger Parameter übergeben wird.

Zusätzlich muß die Anwendung das Element *hwndOwner* in der entsprechenden Datenstruktur für das Dialogfenster setzen. Dieses Element muß das Handle des Fensters enthalten, das die Botschaft *HELPMSGSTRING* erhalten soll.

Die Anwendung kann die Anforderung auch über eine Filterfunktion bearbeiten. Die Filterfunktion erkennt diese Anforderung, wenn der Parameter *wParam* der Botschaft *WM_COMMAND* den Wert *psh15* hat.

Siehe auch *RegisterWindowMessage*

LBSELCHSTRING 3.1

Beschreibung Die Botschaft *LBSELCHSTRING* wird vom Dialogfenster *Datei öffnen* oder *Datei speichern unter* an eine Filterfunktion der Anwendung gesendet, wenn der Anwender in der Liste *Dateiname* eine Auswahl vorgenommen oder verändert hat. Diese Botschaft ermöglicht es einer Anwendung, eine neue Auswahl zu erkennen und anwendungsspezifische Aktionen auszuführen, wie z.B. die Aktualisierung eines anwenderspezifischen Steuerelements im Dialogfenster.

Parameter *wParam:* Bezeichnet die Liste, in der die Auswahl erfolgt ist.

lParam: Bezeichnet das Listenelement und den Typ der Auswahl. Das niederwertige *WORD* des Parameters *lParam* bezeichnet das Listenelement. Das höherwertige *WORD* kann einen der folgenden Werte annehmen:

Wert	Bedeutung
CD_LBSELCHANGE	Bedeutet, daß das Element im niederwertigen *WORD* des Parameters *lParam* ein Element in einer Liste für Einfachauswahl war.
CD_LBSELSUB	Das Element, das durch das niederwertige *WORD* des Parameters *lParam* bezeichnet wird, ist nicht mehr in einer Mehrfachauswahlliste ausgewählt.
CB_LBSELADD	Das Element, das durch das niederwertige *WORD* des Parameters *lParam* bezeichnet wird, wurde in einer Mehrfachauswahlliste ausgewählt.
CD_LBSELNOITEMS	Bedeutet, daß in einer Mehrfachauswahlliste keine Elemente vorhanden sind.

Rückgabewert Die Anwendung liefert Null zurück.

Hinweis Um die Botschaft *LBSELCHSTRING* verwenden zu können, muß die Anwendung einen neuen Botschaftsbezeichner erzeugen. Dies erfolgt durch einen Aufruf der Funktion *RegisterWindowMessage*, in dem die Konstante *LBSELCHSTRING* als einziger Parameter übergeben wird.

Siehe auch *RegisterWindowMessage*

SETRGBSTRING 3.1

Beschreibung Die Botschaft *SETRGBSTRING* wird von der Filterfunktion einer Anwendung an ein Dialogfenster *Farben* gesendet, um eine anwenderdefinierte Farbe zu setzen.

Parameter *wParam:* Wird nicht verwendet.

lParam: Bestimmt die Farbe, die gesetzt werden soll. Dieser Parameter muß einen RGB-Wert enthalten.

Rückgabewert Diese Botschaft hat keinen Rückgabewert.

Hinweis Um die Botschaft *SETRGBSTRING* verwenden zu können, muß die Anwendung einen neuen Botschaftsbezeichner erzeugen. Dies erfolgt durch einen Aufruf der Funktion *RegisterWindowMessage*, in dem die Konstante *SETRGBSTRING* als einziger Parameter übergeben wird.

Siehe auch *RegisterWindowMessage*

SHAREVISTRING 3.1

Beschreibung Die Botschaft *SHAREVISTRING* wird vom Dialogfenster *Datei öffnen* oder *Datei speichern unter* an die Filterfunktion einer Anwendung gesendet, wenn das Dialogfenster eine Datei im Netz öffnen sollte und dabei eine Zugriffsverletzung aufgetreten ist.

Parameter *wParam:* Wird nicht verwendet.

lParam: Zeigt auf einen String, der den Pfad- und Dateinamen bezeichnet, der die Zugriffsverletzung verursacht hat. Dieser String ist das Element *szPathName* der *OFSTRUCT*-Struktur, auf die der zweite Parameter der Funktion *OpenFile* zeigt.

Rückgabewert Der Rückgabewert wird im folgenden »Hinweis« beschrieben.

Hinweis Um die Botschaft *SHAREVISTRING* verwenden zu können, muß die Anwendung einen neuen Botschaftsbezeichner erzeugen. Dies erfolgt durch einen Aufruf der Funktion *RegisterWindowMessage*, in dem die Konstante *SHAREVISTRING* als einziger Parameter übergeben wird.

Diese Botschaft wird von der Funktion *OpenFile* gesendet. Sie wird nicht gesendet, wenn das Flag *OFN_SHAREAWARE* im Element Flags der *OPENFILENAME*-Struktur gesetzt ist.

Wenn die Filterfunktion *SHAREVISTRING* empfängt, sollte sie *OFN_SHAREWARN*, *OFN_SHARENOWARN* oder *OFN_SHAREFALLTHROUGH* zurückgeben. Weitere Informationen über diese Flags finden Sie in der Beschreibung der *OPENFILENAME*-Struktur in Kapitel 7 »Strukturen«.

Siehe auch *OpenFile, RegisterWindowMessage*

14. Botschaften installierbarer Treiber

Die Botschaften installierbarer Treiber benachrichtigen installierbare Treiber über bestimmte Ereignisse, wie z.B. das Laden des Treibers, das Entfernen des Treibers aus dem Speicher oder das Anzeigen eines Dialogfensters für die Konfiguration des Treibers. Microsoft Windows Version 3.1 sendet Botschaften installierbarer Treiber an die Funktion *DriverProc* eines installierbaren Treibers, wenn eine Anwendung Funktionen wie *OpenDriver*, *SendDriverMessage* und *CloseDriver* aufruft.

In diesem Kapitel sind die Botschaften für installierbare Treiber in alphabetischer Reihenfolge aufgeführt.

DRV_CLOSE 3.1

Beschreibung Die Botschaft *DRV_CLOSE* ist die erste Botschaft, die von Windows an einen installierbaren Treiber gesendet wird, nachdem eine Anwendung die Funktion *CloseDriver* aufgerufen hat.

Parameter *dwDriverIdentifier:* Bestimmt den von der Funktion *OpenDriver* zurückgelieferten eindeutigen 32-Bit-Bezeichner.

hDriver: Bezeichnet die Instanz des installierbaren Treibers, die geschlossen werden soll.

lParam1: Bestimmt treiberspezifische Daten.

lParam2: Bestimmt treiberspezifische Daten.

Rückgabewert Ein installierbarer Treiber liefert einen Wert ungleich Null zurück, wenn seine Funktion *DriverProc* den Treiber erfolgreich schließen konnte. Andernfalls liefert er Null zurück.

Hinweis Die Parameter *lParam1* und *lParam2* dieser Botschaft und die Parameter *lParam1* und *lParam2* der Funktion *CloseDriver* bestimmen identische Werte.

Wenn ein Treiber diese Botschaft bearbeitet, muß er seinen privaten Verwendungszähler verringern. Wenn die Variable Null ist, schließt Windows den Treiber.

Siehe auch *DRV_OPEN*

DRV_CONFIGURE 3.1

Beschreibung Die Botschaft *DRV_CONFIGURE* wird gesendet, um einem installierbaren Treiber zu melden, daß er sein privates Konfigurationsdialogfenster anzeigen soll.

Parameter *dwDriverIdentifier:* Bestimmt einen eindeutigen 32-Bit-Wert, der den installierbaren Treiber bezeichnet.

hDriver: Bezeichnet eine Instanz des installierbaren Treibers.

lParam1: Bestimmt das Handle des dem Konfigurationsfenster übergeordneten Fensters. Dieses Handle ist im niederwertigen *WORD* des Parameters gespeichert.

lParam2: Zeigt auf eine optionale *DRVCONFIGINFO*-Struktur. Ein installierbarer Treiber sollte die Gültigkeit dieses Parameters überprüfen, bevor er ihn verwendet.
Diese Struktur hat folgende Form:

```
typedef struct tagDRVCONFIGINFO {
   DWORD   dwDCISize;
   LPCSTR  lpszDCISectionName;
   LPCSTR  lpszDCIAliasName;
} DRVCONFIGINFO;
```

Rückgabewert Ein installierbarer Treiber liefert einen Wert ungleich Null zurück, wenn er diese Botschaft bearbeitet hat. Andernfalls liefert er Null zurück.

Hinweis Ein installierbarer Treiber, der die *DRV_CONFIGURE*-Botschaft unterstützt, muß seine eigene Dialogfenstervorlage und Dialogfensterfunktion bereitstellen. Die Konfigurationsanforderungen des Anwenders müssen in einer entsprechenden Datei aufgezeichnet werden. Diese Datei kann SYSTEM.INI oder eine andere Datei sein, die vom Treiber für diesen Zweck benutzt wird.

Siehe auch *DRV_QUERYCONFIGURE*

DRV_DISABLE 3.1

Beschreibung Die Botschaft *DRV_DISABLE* ist die zweite Botschaft, die von Windows an einen installierbaren Treiber gesendet wird, nachdem eine Anwendung die Funktion *CloseDriver* aufgerufen hat.

Parameter *dwDriverIdentifier:* Wird nicht verwendet.

hDriver: Bezeichnet eine Instanz des installierbaren Treibers.

lParam1: Wird nicht verwendet.

lParam2: Wird nicht verwendet.

Rückgabewert Ein installierbarer Treiber liefert Null zurück, wenn er diese Botschaft bearbeitet hat.

Siehe auch *DRV_CLOSE*

DRV_ENABLE 3.1

Beschreibung Die Botschaft *DRV_ENABLE* wird an einen installierbaren Teiber gesendet, wenn er geladen oder erneut geladen wird. Wenn Windows erneut geladen wird, nachdem zu einer MS-DOS-Anwendung umgeschaltet wurde, wird diese Botschaft ebenfalls gesendet.

Parameter *dwDriverIdentifier:* Wird nicht verwendet.

hDriver: Bezeichnet eine Instanz des installierbaren Treibers.

lParam1: Wird nicht verwendet.

lParam2: Wird nicht verwendet.

Rückgabewert Ein installierbarer Treiber liefert Null zurück, wenn er diese Botschaft bearbeitet hat.

Hinweis Wenn die Funktion *DriverProc* diese Botschaft erhält, sollte sie alle treiberspezifischen Strukturen mit Standardwerten initialisieren.

Siehe auch *DRV_OPEN*

DRV_EXITAPPLICATION 3.1

Beschreibung Die Botschaft *DRV_EXITAPPLICATION* wird an alle installierbaren Treiber gesendet, wenn eine Anwendung beendet wird.

Parameter *dwDriverIdentifier:* Bestimmt einen eindeutigen 32-Bit-Wert, der den installierbaren Treiber bezeichnet.

lParam1: Bestimmt den Typ der Beendigung der Anwendung. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
DRVEA_NORMALEXIT	Ist gesetzt, wenn die Anwendung normal beendet wurde.
DRVEA_ABNORMALEXIT	Ist gesetzt, wenn die Anwendung wegen eines Anwendungs- oder Systemfehlers beendet wurde.

lParam2: Wird nicht verwendet.

Rückgabewert Der von der Anwendung zurückgelieferte Wert wird für diese Botschaft ignoriert.

Siehe auch *DRV_EXITSESSION*

DRV_EXITSESSION 3.1

Beschreibung Die Botschaft *DRV_EXITSESSION* wird an alle installierbaren Treiber gesendet, wenn Windows beendet werden soll.

Parameter *dwDriverIdentifier:* Bestimmt einen eindeutigen 32-Bit-Wert, der den installierbaren Treiber bezeichnet.

lParam1: Reserviert.

lParam2: Reserviert.

Rückgabewert Der von der Anwendung zurückgelieferte Wert wird für diese Botschaft ignoriert.

Hinweis Wenn diese Botschaft gesendet wird, sind die Anwenderschnittstellen und alle anderen Treiber noch aktiviert.

Siehe auch *DRV_EXITAPPLICATION*

DRV_FREE 3.1

Beschreibung Die Botschaft *DRV_FREE* ist die dritte Botschaft, die von Windows an einen installierbaren Treiber gesendet wird, nachdem eine Anwendung die Funktion *CloseDriver* aufgerufen hat.

Parameter *dwDriverIdentifier:* Wird nicht verwendet.

hDriver: Bezeichnet eine Instanz des installierbaren Treibers.

lParam1: Wird nicht verwendet.

lParam2: Wird nicht verwendet.

Rückgabewert Ein installierbarer Treiber liefert Null zurück, wenn er diese Botschaft bearbeitet hat.

Hinweis Wenn die Funktion *DriverProc* eines Treibers diese Botschaft erhält, sollte er den Speicher freigeben, der für alle treiberspezifischen Strukturen reserviert wurde.

DRV_INSTALL 3.1

Beschreibung Die Botschaft *DRV_INSTALL* wird während der Initialisierung eines Treibers an einen installierbaren Treiber gesendet.

Parameter *dwDriverIdentifier:* Bestimmt einen eindeutigen 32-Bit-Wert, der einen installierbaren Treiber bezeichnet.

hDriver: Bezeichnet eine Instanz des installierbaren Treibers.

lParam1: Wird nicht verwendet.

lParam2: Zeigt auf eine optionale *DRVCONFIGINFO*-Struktur. Ein installierbarer Treiber sollte die Gültigkeit dieses Parameters überprüfen, bevor er ihn verwendet.

Die Struktur hat folgende Form:

```
typedef struct tagDRVCONFIGINFO {
   DWORD   dwDCISize;
   LPCSTR  lpszDCISectionName;
   LPCSTR  lpszDCIAliasName;
} DRVCONFIGINFO;
```

Rückgabewert Ein installierbarer Treiber liefert einen Wert ungleich Null zurück, wenn er diese Botschaft bearbeitet hat. Andernfalls liefert er Null zurück.

Hinweis Wenn der Treiber diese Botschaft erhält, erzeugt er für den Treiber einen Eintrag in der Datei SYSTEM.INI und führt weitere nötige Konfigurationsoperationen durch.

DRV_LOAD 3.1

Beschreibung Die Botschaft *DRV_LOAD* wird an einen installierbaren Treiber gesendet, sobald er geladen worden ist.

Parameter *dwDriverIdentifier:* Wird nicht verwendet.

hDriver: Bezeichnet eine Instanz des installierbaren Treibers.

lParam1: Wird nicht verwendet.

lParam2: Wird nicht verwendet.

Rückgabewert Ein installierbarer Treiber liefert einen Wert ungleich Null zurück, wenn seine Funktion *DriverProc* den Treiber erfolgreich geladen hat. Andernfalls liefert er Null zurück.

DRV_OPEN 3.1

Beschreibung Die Botschaft *DRV_OPEN* wird an einen installierbaren Treiber gesendet, wenn er geöffnet wird.

Parameter *dwDriverIdentifier:* Bestimmt einen eindeutigen 32-Bit-Wert, der den installierbaren Treiber bezeichnet.

hDriver: Bezeichnet eine Instanz des installierbaren Treibers.

lParam1: Zeigt auf einen nullterminierten String. Er enthält ASCII-Zeichen, die in der Datei SYSTEM.INI hinter dem Treibernamen stehen.

lParam2: Enthält die Daten, die durch den Parameter *lParam*, das dritte Argument der Funktion *OpenDriver*, bestimmt werden.

Rückgabewert Ein installierbarer Treiber liefert einen Wert ungleich Null zurück, wenn er diese Botschaft bearbeitet hat. Andernfalls liefert er Null zurück.

Hinweis Wenn in der Datei SYSTEM.INI keine Zeichen hinter dem Treibernamen stehen, ist *lParam1* ein Zeiger auf *NULL*.

Siehe auch *DRV_CLOSE*

DRV_POWER 3.1

Beschreibung Die Botschaft *DRV_POWER* wird an einen installierbaren Treiber gesendet, wenn die Stromversorgung des zugeordneten Geräts ein- oder ausgeschaltet werden soll.

Parameter *dwDriverIdentifier:* Bestimmt einen eindeutigen 32-Bit-Wert, der einen installierbaren Treiber bezeichnet.

hDriver: Bezeichnet eine Instanz des installierbaren Treibers.

lParam1: Wird nicht verwendet.

lParam2: Wird nicht verwendet.

Rückgabewert Ein installierbarer Treiber liefert einen Wert ungleich Null zurück, wenn er diese Botschaft bearbeitet hat. Andernfalls liefert er Null zurück.

DRV_QUERYCONFIGURE 3.1

Beschreibung Die Botschaft *DRV_QUERYCONFIGURE* wird an einen installierbaren Treiber gesendet, um festzustellen, ob er vom Anwender konfiguriert werden kann.

Parameter *dwDriverIdentifier:* Bestimmt einen eindeutigen 32-Bit-Wert, der den installierbaren Treiber bezeichnet.

hDriver: Bezeichnet eine Instanz des installierbaren Treibers.

lParam1: Wird nicht verwendet.

lParam2: Wird nicht verwendet.

Rückgabewert Ein installierbarer Treiber liefert einen Wert ungleich Null zurück, wenn er eine anwenderdefinierte Konfiguration unterstützt und ein Dialogfenster zur Konfiguration anzeigen kann. Andernfalls liefert er Null zurück.

Siehe auch *DRV_CONFIGURE*

DRV_REMOVE 3.1

Beschreibung Die Botschaft *DRV_REMOVE* wird von einer Anwendung an einen installierbaren Treiber gesendet, um ihm mitzuteilen, daß er aus dem System entfernt wird.

Parameter *dwDriverIdentifier:* Bestimmt einen eindeutigen 32-Bit-Wert, der den installierbaren Treiber bezeichnet.

lParam1: Nicht verwendet.

lParam2: Nicht verwendet.

Rückgabewert Ein installierbarer Treiber gibt einen Wert ungleich Null zurück, wenn er diese Botschaft bearbeitet. Andernfalls gibt er Null zurück.

Hinweis Sobald ein installierbarer Treiber diese Botschaft erhält, sollte er die entsprechenden Einträge aus der Datei SYSTEM.INI entfernen.

DRV_USER 3.1

Beschreibung Die Botschaft *DRV_USER* ist eine anwenderdefinierte oder treiberabhängige Botschaft.

Parameter *dwDriverIdentifier:* Dieser Parameter ist nicht vordefiniert und der Wert ist treiberabhängig.

hDriver: Dieser Parameter ist nicht vordefiniert und der Wert ist treiberabhängig.

lParam1: Dieser Parameter ist nicht vordefiniert und der Wert ist treiberabhängig.

lParam2: Dieser Parameter ist nicht vordefiniert und der Wert ist treiberabhängig.

Rückgabewert Der Rückgabewert ist vom jeweiligen Treiber abhängig.

15. Grafikdateiformate

Dieses Kapitel beschreibt die Formate von Grafikdateien, die Microsoft Windows verwendet. Grafikdateien sind zum Beispiel Bitmap-Dateien, Symbol- und Mauszeigerressourcendateien.

Dateiformate von Bitmaps

Die Bitmap-Dateien von Windows sind in einem geräteunabhängigen Bitmap-Format (DIB) abgespeichert. Dies erlaubt Windows, das Bitmap auf jeder Art von Anzeigegerät darzustellen. Hierbei bedeutet »geräteunabhängig«, daß das Bitmap die Farbe der Pixel bestimmt, unabhängig von der Methode, die ein bestimmtes Gerät zur Repräsentation von Farben verwendet. Die Standardnamenserweiterung für DIB-Dateien unter Windows ist .BMP.

Strukturen der Bitmap-Dateien

Jede Bitmap-Datei enthält einen Datei-Header, einen Header für die Bitmap-Informationen, eine Farbtabelle und ein Byte-Array, das die Bitmap-Bits definiert. Die Datei hat folgende Form:

```
BITMAPFILEHEADER bmfh;
BITMAPINFOHEADER bmih;
RGBQUAD aColors[];
BYTE aBitmapBits[];
```

Der Header der Bitmap-Datei enthält Informationen über den Typ, die Größe und das Layout einer geräteunabhängigen Bitmap-Datei. Der Header ist als *BITMAPFILEHEADER*-Struktur definiert.

Der Header für die Bitmap-Informationen ist als *BITMAPINFOHEADER*-Struktur definiert. Er bestimmt die Abmessungen, den Komprimierungstyp und das Farbformat für das Bitmap.

Die Farbtabelle ist als Array von *RGBQUAD*-Strukturen definiert und enthält so viele Elemente wie Farben in dem Bitmap vorhanden sind. Die Farbtabelle

steht für Bitmaps mit 24 Farb-Bits nicht zur Verfügung, weil jedes Pixel durch 24-Bit-RGB-Werte (rot-grün-blau) im tatsächlichen Datenbereich des Bitmap dargestellt wird. Die Farben sollten in der Tabelle in der Reihenfolge ihrer Bedeutung erscheinen. Dies hilft einem Bildschirmtreiber, ein Bitmap an ein Gerät zu übergeben, das nicht so viele Farben anzeigen kann, wie in dem Bitmap vorhanden sind. Befindet sich das geräteunabhängige Bitmap im Format der Windows-Version 3.0 oder einer späteren Version, kann der Treiber das Element *biClrImportant* der *BITMAPINFOHEADER*-Struktur verwenden, um festzustellen welche Farben von Bedeutung sind.

Die *BITMAPINFO*-Struktur kann als Repräsentation einer Kombination aus dem Header für die Bitmap-Informationen und der Farbtabelle verwendet werden. Die Bitmap-Bits folgen der Farbtabelle und bestehen aus einem Array von *BYTE*-Werten, die aufeinanderfolgende Zeilen (Scan-Zeilen) des Bitmap darstellen. Jede Scan-Zeile besteht aus aufeinanderfolgenden Byte, die die Pixel in der Scan-Zeile von links nach rechts repräsentieren. Die Anzahl der Byte für eine Scan-Zeile hängt vom Farbformat und der Breite des Bitmap in Pixel ab. Falls notwendig, muß eine Scan-Zeile bis zu einer 32-Bit-Grenze mit Nullen aufgefüllt werden. Das Bitmap kann jedoch an beliebiger Stelle Segmentgrenzen überschreiten. Die Scan-Zeilen in dem Bitmap werden von unten nach oben gespeichert. Dies bedeutet, daß das erste Byte im Array das Pixel in der linken unteren Ecke und das letzte Byte das Pixel in der rechten oberen Ecke des Bitmap repräsentiert.

Das Element *biBitCount* der *BITMAPINFOHEADER*-Struktur legt die Anzahl der Bits fest, die jedes Pixel und die maximale Zahl der Farben in dem Bitmap definieren. Dieses Element kann folgende Werte annehmen:

Wert	Bedeutung
1	Das Bitmap ist einfarbig und die Farbtabelle enthält zwei Einträge. Jedes Bit im Array des Bitmap stellt ein Pixel dar. Ist das Bit gelöscht, wird das Pixel in der Farbe des ersten Eintrags der Farbtabelle angezeigt. Ist das Bit gesetzt, wird das Pixel in der Farbe des zweiten Eintrags der Farbtabelle angezeigt.
4	Das Bitmap besitzt maximal 16 Farben. Jedes Pixel in dem Bitmap wird durch einen 4-Bit-Index in die Farbtabelle dargestellt. Ist das erste Byte in dem Bitmap zum Beispiel 0x1F, repräsentiert das Byte zwei Pixel. Das erste Pixel erhält die Farbe im zweiten Tabelleneintrag, das zweite Pixel die Farbe im sechzehnten Tabelleneintrag.
8	Das Bitmap besitzt maximal 256 Farben. Jedes Pixel in dem Bitmap wird durch einen 1-Byte-Index in die Farbtabelle dargestellt. Ist das erste Byte in dem Bitmap zum Beispiel 0x1F, besitzt das erste Pixel die Farbe des 32. Tabelleneintrags.
24	Das Bitmap besitzt maximal 2^{24} Farben. Das Element *bmiColors* (oder *bmciColors*) ist *NULL* und jede 3-Byte-Sequenz im Array des Bitmap repräsentiert die relativen Intensitäten von Rot, Grün und Blau für ein Pixel.

Das Element *biClrUsed* der *BITMAPINFOHEADER*-Struktur bestimmt die Anzahl der Farbindizes in die Farbtabelle, die tatsächlich von dem Bitmap verwen-

det werden. Ist das Element *biClrUsed* auf Null gesetzt, verwendet das Bitmap die maximale Anzahl von Farben, entsprechend dem Wert des Elements *biBitCount*.

Eine andere Art von Bitmap-Dateien verwendet die *BITMAPCOREINFO*-, *BITMAPCOREHEADER*-, und *RGBTRIPLE*-Strukturen.

Bitmap-Komprimierung

Die Windows-Versionen 3.0 und später unterstützen *run-length*-kodierte Formate (RLE = run-length encoded) zur Komprimierung von Bitmaps, die 4 Bits und 8 Bits pro Pixel verwenden. Eine Komprimierung vermindert den Speicherplatz auf dem Laufwerk und im Speicher, den das Bitmap benötigt.

Komprimierung von Bitmaps mit 8 Bits pro Pixel

Wird das Element *biCompression* der *BITMAPINFOHEADER*-Struktur auf *BI_RLE8* gesetzt, wird das geräteunabhängige Bitmap unter Verwendung eines *run-length*-kodierten Formats für ein Bitmap mit 256 Farben komprimiert. Dieses Format verwendet zwei Modi: den kodierten Modus und den absoluten Modus. Beide Modi können überall innerhalb eines einzelnen Bitmap auftreten.

Kodierter Modus Im kodierten Modus besteht eine Informationseinheit aus zwei Byte. Das erste Byte bestimmt die Anzahl aufeinanderfolgender Pixel, die unter Verwendung des im zweiten Byte enthaltenen Farbindex gezeichnet werden sollen.

Das erste Byte des Paares kann auf Null gesetzt werden, um das Zeilenende, das Ende des Bitmap oder einen Deltawert zu markieren. Diese Interpretation hängt vom Wert des zweiten Byte ab, der sich im Bereich von 0x00 bis 0x02 bewegen muß. Die Werte für das zweite Byte haben folgende Bedeutung:

Zweites Byte	Bedeutung
0	Zeilenende.
1	Ende des Bitmap.
2	Deltawert. Die zwei folgenden Byte enthalten vorzeichenlose Werte, die den horizontalen und vertikalen Offset des nächsten Pixel von der jetztigen Position aus bestimmen.

Absoluter Modus Der absolute Modus wird dadurch angezeigt, daß das erste Byte des Paares auf Null, und das zweite Byte auf einen Wert zwischen 0x03 und 0xFF gesetzt ist. Das zweite Byte stellt die Anzahl der nachfolgenden Byte dar, die alle den Farbindex eines einzigen Pixel enthalten. Jede dieser Byte-Sequenzen muß an einer geraden Adresse ausgerichtet sein.

Unten sehen Sie ein Beispiel für ein *run-length*-kodiertes 8-Bit-Bitmap. Die hexadezimalen Werte in der zweiten Spalte repräsentieren jeweils den Farbindex für ein einzelnes Pixel:

Komprimierte Daten	Entkomprimierte Daten
03 04	04 04 04
05 06	06 06 06 06 06
00 03 45 56 67 00	45 56 67
02 78	78 78
00 02 05 01	5 Pixel nach rechts und 1 Pixel nach unten
02 78	78 78
00 00	Zeilenende
09 1E	1E 1E 1E 1E 1E 1E 1E 1E 1E
00 01	Ende des RLE-Bitmap

Komprimierung von Bitmaps mit 4-Bits pro Pixel

Ist das Element *biCompression* der *BITMAPINFOHEADER*-Struktur auf *BI_RLE4* gesetzt, wird das geräteunabhängige Bitmap unter Verwendung eines *run-length*-kodierten Formats für ein 16-farbiges Bitmap komprimiert.

Kodierter Modus Im kodierten Modus besteht eine Informationseinheit aus zwei Byte. Das erste Byte enthält die Anzahl der Pixel, die unter Verwendung der Farbindizes im zweiten Byte gezeichnet werden sollen.

Das zweite Byte enthält zwei Farbindizes, einen im höherwertigen Nibble des Byte (das sind seine vier niederwertigen Bits) und einen im niederwertigen Nibble. Das erste Pixel wird in der Farbe, die das höherwertige Nibble bestimmt, das zweite in der Farbe des niederwertigen Nibble, das dritte wieder in der Farbe des höherwertigen Nibble gezeichnet. Dies wird fortgesetzt, bis alle Pixel gezeichnet werden, die im ersten Byte angegeben sind.

Das erste Byte des Paares kann auf Null gesetzt werden, um das Zeilenende, das Ende des Bitmap oder einen Deltawert zu markieren. Diese Interpretation hängt vom Wert des zweiten Byte ab, der sich im Bereich von 0x00 bis 0x02 bewegen muß. Die Bedeutung dieser Werte ist dieselbe wie bei 8-Bit-Bitmaps.

Absoluter Modus Im absoluten Modus enthält das erste Byte des Paares den Wert Null, das zweite Byte die Anzahl der nachfolgenden Farbindizes und die darauffolgenden Byte Farbindizes in ihren höher- und niederwertigen Nibbles. Jede Byte-Sequenz muß an einer geraden Adresse ausgerichtet sein.

Unten ein Beispiel für ein *run-length*-kodiertes 4-Bit-Bitmap. Die einstelligen hexadezimalen Werte in der zweiten Spalte repräsentieren jeweils den Farbindex für ein einzelnes Pixel:

Komprimierte Daten	Entkomprimierte Daten
03 04	0 4 0
05 06	0 6 0 6 0
00 06 45 56 67 00	4 5 5 6 6 7
04 78	7 8 7 8
00 02 05 01	5 Pixel nach rechts und 1 Pixel nach unten
04 78	7 8 7 8
00 00	Zeilenende
09 1E	1E 1E 1E 1E 1E 1E 1E 1E 1E
00 01	Ende des RLE-Bitmap

Beispiel für ein Bitmap

Das folgende Beispiel ist die Textrepräsentation eines 16-farbigen Bitmap (4 Bits pro Pixel):

```
Win3DIBFile
                BitmapFileHeader
                     Type       19978
                     Size       3118
                     Reserved1  0
                     Reserved2  0
                     OffsetBits 118
                BitmapInfoHeader
                        Size          40
                        Width         80
                        Height        75
                        Planes        1
                        Bitcount      4
                        Compression   0
                        SizeImage     3000
                        XPelsPerMeter 0
                        YPelsPerMeter 0
                        ColorsUsed    16
                        ColorsImportant 16
                Win3ColorTable
                       Blue  Green  Red   Unused
[00000000]             84    252    84    0
[00000001]             252   252    84    0
[00000002]             84    84     252   0
[00000003]             252   84     252   0
[00000004]             84    252    252   0
[00000005]             252   252    252   0
[00000006]             0     0      0     0
[00000007]             168   0      0     0
[00000008]             0     168    0     0
[00000009]             168   168    0     0
[0000000A]             0     0      168   0
[0000000B]             168   0      168   0
[0000000C]             0     168    168   0
[0000000D]             168   168    168   0
[0000000E]             84    84     84    0
[0000000F]             252   84     84    0
                Bild
    .                           Bitmap-Daten
    .
    .
```

Dateiformat von Symbolressourcen

Die Ressourcendatei eines Symbols enthält Bilddaten für Symbole, die von Windows-Anwendungen verwendet werden. Die Datei besteht aus einem Symbolverzeichnis, das die Anzahl und die Typen der Symbole in der Datei bestimmt und einem oder mehreren Symbolen. Die Standardnamenserweiterung für Symbolressourcen ist .ICO.

Symbol-verzeichnis

Jede Symbolressourcendatei beginnt mit einem Symbolverzeichnis. Das Symbolverzeichnis ist als *ICONDIR*-Struktur definiert und bestimmt die Anzahl der Symbole in der Ressource, die Abmessungen und die Formate aller Symbole. Die *ICONDIR*-Struktur hat folgende Form:

```
typedef struct ICONDIR {
        WORD idReserved;
        WORD idType;
        WORD idCount;
        ICONDIRENTRY idEntries[1];
} ICONHEADER;
```

Die *ICONDIR*-Struktur besitzt die folgenden Elemente:

idReserved: Reserviert, muß Null sein.

idType: Bestimmt den Ressourcentyp. Dieses Element wird auf 1 gesetzt.

idCount: Bestimmt die Anzahl der Verzeichniseinträge.

idEntries: Bestimmt ein Array von *ICONDIRENTRY*-Strukturen, die Informationen über die einzelnen Symbole enthalten. Das Element *idCount* bestimmt die Anzahl der Strukturen im Array.

Die *ICONDIRENTRY*-Struktur bestimmt die Abmessungen und das Farbformat für ein Symbol. Die Struktur besitzt die folgende Form:

```
struct IconDirectoryEntry {
         BYTE bWidth;
         BYTE bHeight;
         BYTE bColorCount;
         BYTE bReserved;
         WORD wPlanes;
         WORD wBitCount;
         DWORD dwBytesInRes;
         DWORD dwImageOffset;
};
```

Die *ICONDIRENTRY*-Struktur besitzt die folgenden Elemente:

bWidth: Bestimmt die Breite des Symbols in Pixel. Zulässig sind die Werte 16, 32 und 64.

bHeight: Bestimmt die Höhe des Symbols in Pixel. Zulässig sind die Werte 16, 32 und 64.

bColorCount: Bestimmt die Anzahl der Farben im Symbol. Zulässig sind die Werte 2, 8 und 16.

bReserved: Reserviert, muß Null sein.

wPlanes: Bestimmt die Anzahl der Farbebenen im Symbol-Bitmap.

wBitCount: Bestimmt die Anzahl der Bits im Symbol-Bitmap.

dwByteInRes: Bestimmt die Größe der Ressource in Byte.

dwImageOffset: Bestimmt den Offset vom Dateianfang bis zum Symbol in Byte.

Symbolbilder

Symbolressourcendateien enthalten für jedes Bild, das im Symbolverzeichnis bezeichnet wird, ein Symbolbild. Ein Symbolbild besteht aus dem Symbolbild-Header, einer Farbtabelle, einer Exklusives-ODER-Maske und einer UND-Maske. Das Symbolbild hat folgende Form:

```
BITMAPINFOHEADER  icHeader;
RGBQUAD           icColors[];
BYTE              icXOR[];
BYTE              icAND[];
```

Der Header des Symbolbilds ist als *BITMAPINFOHEADER*-Struktur definiert und bestimmt Abmessungen und Farbformat des Symbol-Bitmap. Es werden nur die Elemente *biSizeImage* bis *biBitCounts* und *biSize* verwendet. Alle anderen Elemente (wie zum Beispiel *biCompression* und *biClrImportant*) müssen auf Null gesetzt werden.

Die Farbtabelle ist als Array von *RGBQUAD*-Strukturen definiert und bestimmt die Farben, die in der Exklusives-ODER-Maske verwendet werden. Wie bei der Farbtabelle in Bitmap-Dateien legt das Element *biBitCount* im Header des Symbolbilds die Anzahl der Elemente im Array fest.

Die Exklusives-ODER-Maske folgt unmittelbar auf die Farbtabelle. Sie ist ein Array von *BYTE*-Werten, die aufeinanderfolgende Zeilen eines Bitmap repräsentieren. Das Bitmap bestimmt die Gestalt und die Farbe des Symbolbilds. Wie bei den Bitmap-Bits in Bitmap-Dateien werden die Bitmap-Daten in einer Symbolressourcendatei in Scan-Zeilen angeordnet, wobei jedes Byte, je nach Farbformat, ein oder mehr Pixel darstellt. Weitere Informationen finden Sie im Abschnitt »Dateiformate von Bitmaps«.

Die UND-Maske folgt unmittelbar auf die Exklusives-ODER-Maske. Sie ist ein Array von *BYTE*-Werten, die ein einfarbiges Bitmap mit derselben Breite und

Höhe wie die Exklusives-ODER-Maske repräsentiert. Das Array ist in Scan-Zeilen angeordnet, wobei jedes Byte für 8 Pixel steht.

Wenn Windows ein Symbol zeichnet, verwendet es die UND- und Exklusives-ODER-Masken, um das Symbolbild mit den bereits auf der Anzeige gezeichneten Pixel zu verbinden. Windows wendet zuerst die UND-Maske mit bitweisen UND-Operationen an. So bleiben bestimmte Pixelfarben erhalten, andere werden entfernt. Danach wendet Windows die Exklusives-ODER-Maske mit bitweisen Exklusives-ODER-Operationen an. Dies bestimmt die endgültige Farbe für jedes Pixel.

Symbolauswahl in Windows

Windows bestimmt die Auflösung der aktuellen Anzeige und paßt sie an die Breite und die Höhe an, die für jede Version eines Symbolbilds angegeben ist. Stellt Windows fest, daß ein Symbolbild und das aktuelle Ausgabegerät genau zueinanderpassen, verwendet es dieses Bild. Andernfalls wird das ähnlichste Maß ausgewählt und das Bild auf die richtige Größe gestreckt.

Enthält die Symbolressourcendatei mehr als ein Bild für eine bestimmte Auflösung, wird das Symbolbild verwendet, das den Farbmöglichkeiten der aktuellen Anzeige am besten entspricht. Entspricht kein Bild genau den Möglichkeiten des Geräts, wird das Bild gewählt, das die größte Zahl an Farben aufweist, ohne die Zahl der Bildschirmfarben zu überschreiten. Überschreiten alle Bilder die Farbmöglichkeiten des Geräts, verwendet Windows das Symbolbild mit den wenigsten Farben.

Dateiformate von Mauszeigerressourcen

Die Datei einer Mauszeigerressource enthält Bilddaten für von Windows-Anwendungen benutzte Mauszeiger. Sie besteht aus einem Mauszeigerverzeichnis, das die Anzahl und den Typ der Mauszeigerbilder in der Datei bezeichnet. Außerdem enthält sie ein oder mehrere Mauszeigerbilder. Die Standardnamenserweiterung für Mauszeigerressourcen ist .CUR.

Das Mauszeigerverzeichnis

Eine Mauszeigerressourcendatei beginnt mit dem Mauszeigerverzeichnis. Das Verzeichnis ist als *CURSORDIR*-Struktur definiert und bestimmt die Anzahl der Mauszeiger in der Datei sowie Abmessungen und Farbformate der einzelnen Bilder. Die *CURSORDIR*-Struktur besitzt folgende Form:

```
typedef struct _CURSORDIR {
   WORD cdReserved;
   WORD cdType;
   WORD cdCount;
   CURSORDIRENTRY cdEntries[];
} CURSORDIR;
```

Die *CURSORDIR*-Struktur besitzt folgende Elemente:

cdReserved: Reserviert, muß Null sein.

cdType: Bestimmt den Ressourcentyp. Dieses Element muß auf 2 gesetzt sein.

cdCount: Bestimmt die Anzahl der Mauszeiger in der Datei.

cdEntries: Bestimmt ein Array von *CURSORDIRENTRY*-Strukturen, die Informationen über einzelne Mauszeiger enthalten. Das Element *cdCount* bestimmt die Anzahl der Strukturen im Array.

Eine *CURSORDIRENTRY*-Struktur bestimmt die Abmessungen und das Farbformat des Mauszeigerbilds.
Die Struktur besitzt die folgende Form:

```
typedef struct _CURSORDIRENTRY {
   BYTE bWidth;
   BYTE bHeight;
   BYTE bColorCount;
   BYTE bReserved;
   WORD wXHotspot;
   WORD wYHotspot;
   DWORD lBytesInRes;
   DWORD dwImageOffset;
} CURSORDIRENTRY;
```

Die *CURSORDIRENTRY*-Struktur besitzt folgende Elemente:

bWidth: Bestimmt die Breite des Mauszeigers in Pixel.

bHeight: Bestimmt die Höhe des Mauszeigers in Pixel.

bColorCount: Reserviert, muß Null sein.

bReserved: Reserviert, muß Null sein.

wXHotspot: Bestimmt die x-Koordinate des Kontaktpunkts in Pixel.

wYHotspot: Bestimmt die y-Koordinate des Kontaktpunkts in Pixel.

lByteInRes: Bestimmt die Größe der Ressource in Byte.

dwImageOffset: Bestimmt den Offset in Byte, vom Dateianfang bis zum Mauszeigerbild gerechnet.

Das Mauszeigerbild

Mauszeigerressourcendateien enthalten für jeden Eintrag im Mauszeigerverzeichnis ein Mauszeigerbild. Ein Mauszeigerbild besteht aus dem Header des Bilds, einer Farbtabelle, einer Exklusives-ODER-Maske und einer UND-Maske. Das Bild besitzt die folgende Form:

```
BITMAPINFOHEADER  crHeader;
RGBQUAD           crColors[];
BYTE              crXOR[];
BYTE              crAND[];
```

Der Kontaktpunkt des Mauszeigers ist ein einzelnes Pixel im Mauszeiger-Bitmap, das Windows zum Nachführen des Mauszeigers verwendet. Die Elemente *wXHotspot* und *wYHotspot* der *CURSORDIRENTRY*-Struktur bestimmen die x- und y-Koordinaten des Kontaktpunkts. Diese Koordinaten sind 16-Bit-Integer-Werte.

Der Header des Mauszeigerbilds ist als *BITMAPINFOHEADER*-Struktur definiert und bestimmt Abmessungen und Farbformat des Mauszeiger-Bitmap. Es werden nur die Elemente *biSizeImage* bis *biBitCount* und *biSize* verwendet. Das Element *biHeight* bestimmt die Höhe des Mauszeigers als Kombination aus der Exklusives-ODER- und der UND-Maske. Die Elemente *biPlanes* und *biBitCount* müssen auf 1 gesetzt sein. Alle anderen Elemente (wie z.B. *biCompression* und *biClrImportant*) müssen auf Null gesetzt werden.

Die Farbtabelle ist als Array von *RGBQUAD*-Strukturen definiert und bestimmt die Farben, die in den Exklusives-ODER-Masken verwendet werden. Für ein Mauszeigerbild enthält die Tabelle genau zwei Strukturen, da das Element *biBitCount* im Header des Bilds immer gleich 1 ist.

Die Exklusives-ODER-Maske folgt unmittelbar auf die Farbtabelle. Sie ist ein Array von *BYTE*-Werten, die aufeinanderfolgende Zeilen eines Bitmap darstellen. Das Bitmap bestimmt die Gestalt und die Farbe des Mauszeigerbilds. Wie bei den Bitmap-Bits in Bitmap-Dateien werden die Bitmap-Daten in der Datei einer Mauszeigerressource in Scan-Zeilen angeordnet, wobei jedes Byte, je nach Farbformat, ein oder mehrere Pixel darstellt. Weitere Informationen finden Sie im Abschnitt »Dateiformate von Bitmaps«.

Die UND-Maske folgt unmittelbar auf die Exklusives-ODER-Maske. Sie ist ein Array von *BYTE*-Werten, die ein monochromes Bitmap mit derselben Breite und Höhe wie die Exklusives-ODER-Maske repräsentieren. Das Array ist in Scan-Zeilen angeordnet, wobei jedes Byte für 8 Pixel steht.

Wenn Windows einen Mauszeiger zeichnet, verwendet es die UND- und Exklusives-ODER-Masken, um das Mauszeigerbild mit den bereits auf der Anzeige gezeichneten Pixel zu verbinden. Windows wendet zuerst die UND-Maske mit bitweisen UND-Operationen an. Dadurch werden die vorhandenen Pixelfarben erhalten oder entfernt. Danach wendet Windows die Exklusives-ODER-Maske mit bitweisen Exklusives-ODER-Operationen an. Dies bestimmt die endgültige Farbe für jedes Pixel.

Um schwarze, weiße, invertierte oder transparente Mauszeiger zu erhalten, müssen die Bit-Masken folgende Werte enthalten:

Ergebnis	UND-Maske	Exklusives-ODER-Maske
Schwarz	0	0
Weiß	0	1
Transparent	1	0
Invertiert	1	1

Mauszeigerauswahl in Windows

Enthält eine Mauszeigerressourcendatei mehr als ein Bild, bestimmt Windows den einer bestimmten Anzeige weitestgehend entsprechenden Mauszeiger, indem Breite und Höhe der Mauszeigerbilder untersucht werden.

16. Das Dateiformat der Zwischenablage

Die Zwischenablage von Microsoft Windows (CLIPBRD.EXE) schreibt und liest ihre Daten in und aus Dateien mit der Namenserweiterung .CLP. Eine Datei mit der Namenserweiterung .CLP enthält einen Wert, der sie als Datendatei der Zwischenablage identifiziert.

Sie enthält außerdem eine oder mehrere Strukturen, die das Format, die Größe und die Position der Daten definieren sowie einen oder mehrere Blöcke mit den eigentlichen Daten.

Header der Zwischenablagedatei

Am Anfang der Zwischenablagedatei steht ein Header, der aus zwei Elementen besteht. Die Elemente dieses Header sind im folgenden aufgeführt:

FileIdentifier: Identifiziert die Datei als Zwischenablagedatei. Dieses Element ist ein 2-Byte-Wert und muß auf *CLIP_ID* gesetzt sein und.

FormatCount: Bestimmt die Anzahl der in der Datei enthaltenen Zwischenablageformate. Dieses Element ist ein 2-Byte-Wert.

Struktur der Zwischenablagedatei

Nach dem Header folgen eine oder mehrere Strukturen. Jede dieser Strukturen bestimmt das Format, die Größe und den Offset eines Blocks, der Zwischenablagedaten enthält.

Im folgenden sind die Elemente dieser Struktur aufgeführt:

FormatID: Bestimmt den Bezeichner des Zwischenablageformats. Dieses Element ist ein 2-Byte-Wert.

LenData: Gibt die Länge der Zwischenablagedaten in Byte an. Es handelt sich um einen 4-Byte-Wert.

OffData: Gibt den Offset des Blocks mit den Zwischenablagedaten in Byte an. Der Offset wird ausgehend vom Anfang der Datei berechnet. Es handelt sich um einen 4-Byte-Wert.

Name: Bezeichnet ein Array mit 79 Zeichen Länge, das den Formatnamen eines privaten Zwischenablageformats angibt.

Nach der letzten dieser Strukturen folgt der erste Block mit Zwischenablagedaten. Bei Bitmaps und Metafiles sind die Daten nach dem Bitmap-Header bzw. den *METAFILEPICT*-Strukturen abgelegt.

17. Das Metafile-Format

Ein Metafile für Microsoft Windows besteht aus einer Reihe von Funktionen der grafischen Geräteschnittstelle (GDI), die ein Bild beschreiben. Metafiles benötigen weniger Platz und sind geräteunabhängiger als Bitmaps. Deshalb können Bilder, die wiederholt in Anwendungen verwendet werden oder die innerhalb der Anwendungen bewegt werden müssen, effizienter gespeichert werden.

Um ein Metafile zu erstellen, erzeugt eine Windows-Anwendung einen speziellen Gerätekontext, der GDI-Befehle zum Speichern der Daten in einer Datei oder im Speicher sendet. Die Anwendung kann das Metafile später abspielen und das Bild anzeigen lassen.

Während des Abspielens zerlegt Windows das Metafile in Records und bezeichnet jedes Objekt mit einem Index in eine Handle-Tabelle. Wird während eines Abspielvorgangs ein *META_DELETEOBJECT*-Objekt gefunden, wird das zugeordnete Objekt aus der Handle-Tabelle entfernt. Der zugehörige Eintrag wird vom nächsten Objekt, das vom Metafile erzeugt wird, wiederverwendet. Eine Anwendung, die Records explizit verändert oder ihr eigenes Metafile erzeugt, sollte die Handle-Tabelle in gleicher Weise verwalten, um die Kompatibilität zu gewährleisten.

In einigen Fällen gibt es zwei Varianten eines Metafile-Record. Eine Variante stellt den Record dar, der unter den Versionen vor 3.0, die andere den Record, der unter der Version 3.0 oder höher erzeugt wurde. Windows-Versionen vor 3.0 zeigen keine Metafiles an, die von Versionen 3.0 und höher aufgezeichnet wurden.

Ein Metafile besteht aus zwei Teilen: einem Header und einer Record-Liste. Der Header und die Records (sowohl typische als auch funktionsspezifische Records) werden in den folgenden drei Abschnitten dieses Kapitels beschrieben.

Metafile-Header

Der Metafile-Header enthält eine Beschreibung der Größe des Metafile und die Anzahl der Zeichenobjekte, die es benutzt. Die Zeichenobjekte können Stifte, Pinsel, Bitmaps oder Schriften sein. Der Metafile-Header hat folgende Form:

```
typedef struct tagMETAHEADER {
  WORD  mtType;
  WORD  mtHeaderSize;
  WORD  mtVersion;
  DWORD mtSize;
  WORD  mtNoObjects;
  DWORD mtMaxRecords;
  WORD  mtNoParameters;
} METAHEADER;
```

Im folgenden sind die Elemente des Metafile-Header beschrieben:

mtType: Gibt an, ob das Metafile im Speicher abgelegt oder in einer Datei aufgezeichnet ist. Dieses Element kann folgende Werte annehmen:

Wert	Bedeutung
0	Das Metafile ist im Speicher abgelegt.
1	Das Metafile ist in einer Datei abgelegt.

mtHeaderSize: Bestimmt die Größe des Metafile-Header in *WORD*.

mtVersion: Bestimmt die Windows-Versionsnummer. Die Versionsnummer 0x300 steht für Windows 3.0 und höher.

mtSize: Bestimmt die Größe der Datei in *WORD*.

mtNoObjects: Gibt die maximale Anzahl der Objekte an, die gleichzeitig im Metafile existieren können.

mtMaxRecord: Gibt die Größe des längsten Record des Metafile in *WORD* an.

mtNoParameters: Wird nicht verwendet.

Standard-Metafile-Record

Die grafische Geräteschnittstelle speichert die meisten GDI-Funktionen, die eine Anwendung zur Erzeugung von Metafiles benutzen kann, in Standard-Records. Ein Standard-Metafile-Record hat folgende Form:

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Im folgenden sind die Elemente eines Standard-Metafile-Record beschrieben:

rdSize: Bestimmt die Größe des Record in *WORD*.

rdFunction: Bestimmt die Funktionsnummer. Dieser Wert kann die Nummer einer Funktion aus der am Ende dieses Abschnitts aufgeführten Tabelle sein.

rdParm: Bezeichnet ein *WORD*-Array, das die Funktionsparameter enthält. Die Funktionsparameter sind im Array in der umgekehrten Reihenfolge aufgelistet, in der sie an die Funktion übergeben werden.

Im folgenden sind die GDI-Funktionen, die in Standard-Records erscheinen, zusammen mit ihren hexadezimalen Werten aufgeführt:

GDI-Funktion	Wert
Arc	0x0817
Chord	0x0830
Ellipse	0x0418
ExcludeClipRect	0x0415
FloodFill	0x0419
IntersectClipRect	0x0416
LineTo	0x0213
MoveTo	0x0214
OffsetClipRgn	0x0220
OffsetViewportOrg	0x0211
OffsetWindowOrg	0x020F
PatBlt	0x061D
Pie	0x081A
RealizePalette (3.0 und höher)	0x0035
Rectangle	0x041B
ResizePalette (3.0 und höher)	0x0139
RestoreDC	0x0127
RoundRect	0x061C
SaveDC	0x001E
ScaleViewportExt	0x0412
ScaleWindowExt	0x0400
SetBkColor	0x0201
SetBkMode	0x0102
SetMapMode	0x0103
SetMapperFlags	0x0231
SetPixel	0x041F
SetPolyFillMode	0x0106
SetROP2	0x0104
SetStretchBltMode	0x0107
SetTextAlign	0x012E
SetTextCharExtra	0x0108
SetTextColor	0x0209
SetTextJustification	0x020A
SetViewportExt	0x020E
SetViewportOrg	0x020D
SetWindowExt	0x020C
SetWindowOrg	0x020B

Positionierbare Windows-Metafiles

Ein positionierbares Windows-Metafile ist ein Standard-Windows-Metafile mit einem zusätzlichen 22-Byte-Header. Der Header enthält Informationen über das Seitenverhältnis und die Originalgröße des Metafile. Diese Informationen erlauben einer Anwendung, das Metafile in der beabsichtigten Form anzuzeigen. Der Header eines positionierbaren Windows-Metafile hat folgende Form:

```
typedef struct {
   DWORD key;
   HANDLE hmf;
   RECT bbox;
   WORD inch;
   DWORD reserved;
   WORD checksum;
} METAFILEHEADER;
```

Im folgenden sind die Elemente des Header eines positionierbaren Metafile aufgeführt:

key: Bestimmt den Binärschlüssel, der diesen Dateityp eindeutig identifiziert. Dieses Element muß auf 0x9AC6CDD7L gesetzt werden.

hmf: Wird nicht verwendet und muß auf Null gesetzt sein.

bbox: Bestimmt die Koordinaten des kleinsten umgebenden Rechtecks. Die Koordinaten sind in den im Element *inch* definierten Metafile-Einheiten angegeben.

inch: Bestimmt die Anzahl der Metafile-Einheiten pro Zoll. Dieser Wert sollte kleiner als 1440 sein, um einen numerischen Überlauf zu vermeiden. Die meisten Anwendungen verwenden die Werte 576 oder 1000.

reserved: Wird nicht verwendet und muß auf Null gesetzt sein.

checksum: Gibt die Prüfsumme an. Diese Summe wird unter Verwendung des Exklusives-ODER-Operators aus den ersten 10 *WORD*-Werten des Header gebildet.

Der tatsächliche Inhalt des Windows-Metafile folgt unmittelbar nach dem Header. Das Format dieses Inhalts ist identisch mit dem Format der Standard-Windows-Metafiles. Bei einigen Anwendungen darf ein positionierbares Windows-Metafile eine Größe von 64 KByte nicht überschreiten.

Anmerkung: Positionierbare Windows-Metafiles sind zur Funktion *GetMetaFile* nicht kompatibel. Anwendungen, die die Metafile-Funktionen zum Lesen und Abspielen von positionierbaren Metafiles verwenden wollen, müssen die Datei mit einer gewöhnlichen Eingabefunktion (wie z.B. *_lread*) lesen. Die Anwendung muß dann den 22-Byte-Header entfernen und ein Standard-Windows-Meafile erzeugen. Dies erfolgt durch einen Aufruf der Funktion *SetMetaFileBits*, wobei die restlichen Byte verwendet werden.

Richtlinien für Windows-Metafiles

Damit Metafiles zwischen verschiedenen Computern und Anwendungen ausgetauscht werden können, sollte jede Anwendung sicherstellen, daß das erzeugte Metafile geräteunabhängig ist und in der Größe angepaßt werden kann. Die folgenden Richtlinien stellen sicher, daß jedes Metafile von anderen Anwendungen benutzt und verändert werden kann:

- In einem der ersten Records sollte ein Abbildungsmodus angegeben sein. Viele Anwendungen, OLE-Anwendungen eingeschlossen, akzeptieren Metafiles nur im *MM_ANISOTROPIC*-Modus.
- Rufen Sie die Funktionen *SetWindowOrg* und *SetWindowExt* auf. Wenn der Anwender die Möglichkeit haben soll, die Abmessungen nachträglich zu verändern, dürfen die Funktionen *SetViewportExt* oder *SetViewportOrg* nicht aufgerufen werden.
- Verwenden Sie den Drucker-Escape-Code *MFCOMMENT*, um Kommentare in das Metafile einzufügen.
- Verwenden Sie in erster Linie die im Abschnitt »Standard-Metafile-Record« aufgeführten Funktionen. Beachten Sie beim Gebrauch der Funktionen folgende Einschränkungen:
  - Verwenden Sie keine Funktionen, die Daten abrufen.
  - Verwenden Sie keine Region-Funktionen. Diese sind geräteabhängig.
  - Verwenden Sie *StretchBlt* oder *StretchDIB* anstelle von *BitBlt*.

Beispiel einer Metafile-Programm-Ausgabe

In diesem Abschnitt wird ein Beispielprogramm und das vom Programm erzeugte Metafile beschrieben. Das Beispielprogramm erzeugt ein kleines Metafile, das ein purpurrotes Rechteck mit grünem Rand zeichnet und die Worte »Hello People« in das Rechteck schreibt.

```
MakeAMetaFile(hDC)
HDC hDC;
{
  HPEN hMetaGreenPen;
  HPRUSH hMetaVioletBrush;
  HDC hDCMeta;
  HANDLE hMeta;

  /* Ein Metafile wird erzeugt. Die Ausgabe erfolgt auf Platte.  */

  hDCMeta = CreateMetaFile((LPSTR) "sample.met");

  hMetaGreenPen = CreatePen(0, 0, (DWORD) 0x0000FF00);
  SelectObject(hDCMeta, hMetaGreenPen);
```

```
hMetaVioletBrush = CreateSolidBrush((DWORD) 0x00FF00FF);
SelectObject(hDCMeta, hMetaVioletBrush);

Rectangle(hDCMeta, 0, 0, 150, 70);

TextOut(hDCMeta, 10, 10, (LPSTR) "Hello People", 12);

/*  Das Metafile ist fertig.  */

hMeta = CloseMetaFile(hDCMeta);

/*  Das soeben erzeugte Metafile wird abgespielt.  */

PlayMetaFile(hDC, hMeta);
```

Das resultierende Metafile SAMPLE.MET besteht aus einem Header und sechs Records. Es ist in der folgenden Tabelle dargestellt:

Binärdaten im Hexadezimalformat	Name des Strukturelements	Bedeutung
0001	*mtType*	Metafile-Header
0009	*mtHeaderSize*	
0300	*mtVersion*	
0000 0036	*mtSize*	
0002	*mtNoObjekts*	
0000 000C	*mtMaxRecords*	
0000	*mtNoParameters*	
0000 0008	*rdSize*	Record Nr. 0
02FA	*rdFunction*	*CreatePenIndirect*
0000 0000 0000 0000 FF00	*rdParm*	*LOGPEN*-Struktur, die den Stift definiert
0000 0004	*rdSize*	Record Nr. 1
012D	*rdFunction*	*SelectObject*
0000	*rdParm*	Objekt Nr. 0, der obige Stift
0000 0007	*rdSize*	Record Nr. 2
02FC	*rdFunction*	*CreateBrushIndirect*
0000 00FF 00FF 0000	*rdParm*	*LOGBRUSH*-Struktur, die den Pinsel definiert
0000 0004	*rdSize*	Record Nr. 3
012D	*rdFunction*	*SelectObject*
0001	*rdParm*	Objekt Nr. 1, der Pinsel
0000 0007	*rdSize*	Record Nr. 4
041B	*rdFunction*	*Rectangle*
0046 0096 0000 0000	*rdParm*	Parameter für *Rectangle* in umgekehrter Reihenfolge

Binärdaten im Hexadezimalformat	Name des Strukturelements	Bedeutung
0000 000C	*rdSize*	Record Nr. 5
0521	*rdFunction*	*TextOut*
000C	*rdParm*	String-Länge
48 65 6C 6C		"Hello People"
6F 20 50 65		
6F 70 6C 65		
000A		y-Wert
000A		x-Wert

Funktionsspezifische Metafile-Records

Die grafische Geräteschnittstelle speichert die meisten GDI-Funktionen zum Erzeugen von Metafiles in Standard-Records. Die restlichen Funktionen sind in funktionsspezifischen Records gespeichert, die im Element *rdParm* Strukturen enthalten. Die Definitionen dieser Records werden in diesem Abschnitt beschrieben.

AnimatePalette

Definition

```
struct {
   DWORD rdSize;
   WORD rdFunction;
   WORD rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x0436.

rdParm: Enthält die folgenden Elemente:

Element	Beschreibung
start	Erster Eintrag, der verändert wird.
numentries	Anzahl der Einträge, die verändert werden sollen.
entries	*PALETTEENTRY*-Blöcke.

BitBlt

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
```

```
   WORD  rdParm[];
}
```

Beschreibung Dieser von Windows-Versionen vor der Version 3.0 gespeicherte Record für *BitBlt* enthält ein geräteabhängiges Bitmap, das nicht auf allen Ausgabegeräten korrekt wiedergegeben wird.

Elemente *rdSize:* Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x0922.

rdParm: Enthält die folgenden Elemente:

Element	Beschreibung
rasterop	Höherwertiges *WORD* der Raster-Operation.
SY	y-Koordinate des Quellursprungs.
SX	x-Koordinate des Quellursprungs.
DYE	y-Abmesssung des Ziels.
DXE	x-Abmessung des Ziels.
DY	y-Koordinate des Zielursprungs.
DX	x-Koordinate des Zielursprungs.
bmWidth	Breite des Bitmap in Pixel.
bmHeight	Höhe des Bitmap in Rasterzeilen.
bmWidthBytes	Anzahl der Byte in jeder Rasterzeile.
bmPlanes	Anzahl der Farbebenen in dem Bitmap.
bmBitsPixel	Anzahl der nebeneinanderliegenden Farb-Bits.
bits	Die geräteabhängigen Bits des Bitmap.

BitBlt

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Beschreibung Dieser für *BitBlt* gespeicherte Record enthält ein geräteunabhängiges Bitmap, das auf jedem Gerät abgespielt werden kann.

Elemente *rdSize:* Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x0940.

rdParm: Enthält die folgenden Elemente:

Element	Beschreibung
raster op	Höherwertiges *WORD* der Raster-Operation.
SY	y-Koordinate des Quellursprungs.
SX	x-Koordinate des Quellursprungs.

Element	Beschreibung
DYE	y-Abmessung des Ziels.
DXE	x-Abmessung des Ziels.
DY	y-Koordinate des Zielursprungs.
DX	x-Koordinate des Zielursprungs.
BitmapInfo	*BITMAPINFO*-Struktur.
bits	Die geräteunabhängigen Bits des Bitmap.

CreateBrushIndirect

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x02FC.

rdParm: Bestimmt den virtuellen Pinsel.

CreateFontIndirect

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x02FB.

rdParm: Bestimmt die virtuelle Schrift.

CreatePalette

Definition

```
struct {
   DWORD       rdSize;
   WORD        rdFunction;
   LOGPALETTE rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x00F7.

rdParm: Bestimmt die virtuelle Palette.

CreatePatternBrush

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Beschreibung Dieser für *CreatePatternBrush* gespeicherte Record enthält ein geräteabhängiges Bitmap, das nicht auf allen Geräten korrekt wiedergegeben wird.

Elemente *rdSize:* Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x01F9.

rdParm: Enthält die folgenden Elemente:

Element	Beschreibung
bmWidth	Breite des Bitmap.
bmHeight	Höhe des Bitmap.
bmWidthBytes	Byte je Rasterzeile.
bmPlanes	Anzahl der Farbebenen.
bmBitsPixel.	Anzahl der benachbarten Farb-Bits, die ein Pixel definieren.
bmBits	Zeiger auf Bit-Werte.
bits	Die geräteabhängigen Bits des Musters.

CreatePatternBrush

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Beschreibung Dieser für *CreatePatternBrush* gespeicherte Record enthält ein geräteunabhängiges Bitmap, das auf allen Geräten abgespielt werden kann.

Elemente *rdSize:* Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x0142.

rdParm: Enthält die folgenden Elemente:

Element	Beschreibung
type	Typ des Bitmap. Dieses Element muß einen der folgenden Werte annehmen:

Element	Beschreibung
	BS_PATTERN: Der Pinsel ist durch einen Aufruf der Funktion *CreatePatternBrush* durch ein geräteabhängiges Bitmap definiert. *BS_DIBPATTERN:* Der Pinsel ist durch einen Aufruf der Funktion *CreateDIBPatternBrush* durch ein geräteunabhängiges Bitmap definiert.
wUsage	Typ der Farbtabelle. Dieses Element gibt an, ob das Element *bmiColors* der *BITMAPINFO*-Struktur explizite RGB-Werte oder Indizes in die aktuell realisierte virtuelle Palette enthält. Dieses Element muß einen der folgenden Werte annehmen: *DIB_RGB_COLORS:* Die Farbtabelle enthält direkt zu übernehmende RGB-Werte. *DIB_PAL_COLORS:* Die Farbtabelle besteht aus einem Index-Array, dessen Indizes in die aktuell realisierte virtuelle Palette verweisen.
bmi	*BITMAPINFO*-Struktur.
bits	Die geräteunabhängigen Bits des Bitmap.

CreatePenIndirect

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x02FA.

rdParm: Bestimmt den virtuellen Stift.

CreateRegion

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x06FF.

rdParm: Gibt die Region an, die erzeugt werden soll.

DeleteObject

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x01F0.

rdParm: Gibt den Index in die Handle-Tabelle für das zu löschende Objekt an.

Escape

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x0626.

rdParm: Enthält folgende Elemente:

Element	Beschreibung
escape_number	Nummer, die den privaten Escape-Code bezeichnet.
count	Anzahl der Informations-Byte.
input_data	Feld variabler Länge. Das Element hat eine Länge von $((count+1) >> 1)$ *WORD*.

ExtTextOut

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x0A36.

rdParm: Enthält folgende Elemente:

Element	Beschreibung
y	Virtueller y-Wert des Startpunkts des String.
x	Virtueller x-Wert des Startpunkts des String.
count	Länge des String.
options	Typ des Rechtecks. Um festzulegen, ob in diesem Element die *ETO_CLIPPED*- oder die *ETO_OPAQUE*-Bits gesetzt sind, sollte eine Anwendung den UND-Operator (&) verwenden. Es ist ungünstig, in diesem Fall den Gleichheitsoperator (==) zu verwenden, weil manche Anwendungen im Parameter *wOptions* der rechteckigen Region, in die die Funktion *ExtTextOut* den Text schreibt, zusätzliche Bits setzen.
rectangle	*RECT*-Struktur. Sie legt die rechteckige Region fest, in die die Funktion *ExtTextOut* Text schreibt. Wenn das Element *options* den Wert Null hat, ist dieses Element nicht vorhanden.
string	Byte-Array, das den String enthält. Das Array hat die Länge ((*count* +1) >>1) *WORD*.
dxarray	Optionales *WORD*-Array, das die Zeichenabstände enthält.

Polygon

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x0324.

rdParm: Enthält folgende Elemente:

Element	Beschreibung
count	Anzahl der Punkte.
list_of_points	Liste der Einzelpunkte.

PolyPolygon

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x0538.

rdParm: Enthält folgende Elemente:

Element	Beschreibung
count	Gesamtzahl der Polygone.
list_of_polygon_counts	Liste der Anzahl der Punkte in jedem Polygon.
list_of_points	Liste der Einzelpunkte.

Polyline

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x0325.

rdParm: Enthält folgende Elemente:

Wert	Beschreibung
count	Anzahl der Punkte.
list_of_points	Liste der Einzelpunkte.

SelectClipRgn

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x012C.

rdParm: Gibt den Index in die Handle-Tabelle für die ausgewählte Region an.

SelectObject

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x012D.

rdParm: Gibt den Index in die Handle-Tabelle für das ausgewählte Objekt an.

SelectPalette

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x0234.

rdParm: Gibt den Index in die Handle-Tabelle für die ausgewählte virtuelle Palette an.

SetDIBitsToDevice

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x0D33.

rdParm: Enthält folgende Elemente:

Element	Beschreibung
wUsage	Flag, das anzeigt, ob die Bitmap-Farbtabelle RGB-Werte oder Indizes in die aktuell realisierte virtuelle Palette enthält.
numscans	Anzahl der Rasterzeilen in dem Bitmap.
startscan	Erste Rasterzeile in dem Bitmap.
srcY	y-Koordinate des Ursprungs des Quellrechtecks des Bitmap.
srcX	x-Koordinate des Ursprungs des Quellrechtecks des Bitmap.
extY	Höhe des Quellrechtecks des Bitmap.
extX	Breite des Quellrechtecks des Bitmap.
destY	y-Koordinate des Ursprungs des Zielrechtecks.
destX	x-Koordinate des Ursprungs des Zielrechtecks.
BitmapInfo	*BITMAPINFO*-Struktur.
bits	Die geräteunabhängigen Bits des Bitmap.

SetPaletteEntries

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x0037.

rdParm: Enthält folgende Elemente:

Element	Beschreibung
start	Erster Eintrag, der in der Palette gesetzt werden muß.
numentries	Anzahl der Einträge, die in der Palette gesetzt werden sollen.
entries	*PALETTEENTRY*-Struktur.

StretchBlt

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Beschreibung Dieser für *StretchBlt* gespeicherte Record enthält ein geräteabhängiges Bitmap, das möglicherweise nicht auf allen Geräten korrekt wiedergegeben wird.

Elemente *rdSize:* Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x0B23.

rdParm: Enthält folgende Elemente:

Element	Beschreibung
raster op	Niederwertiges *WORD* der Raster-Operation.
raster op	Höherwertiges *WORD* der Raster-Operation.
SYE	y-Abmessung der Quelle.
SXE	x-Abmessung der Quelle.
SY	y-Koordinate des Ursprungs der Quelle.
SX	x-Koordinate des Ursprungs der Quelle.
DYE	y-Abmessung des Ziels.
DXE	x-Abmessung des Ziels.
DY	y-Koordinate des Ursprungs des Ziels.
DX	x-Koordinate des Ursprungs des Ziels.
bmWidth	Breite des Bitmap in Pixel.
bmHeight	Höhe des Bitmap in Rasterzeilen.
bmWidthBytes	Anzahl der Byte in jeder Rasterzeile.
bmPlanes	Anzahl der Farbebenen in dem Bitmap.
bmBitsPixel	Anzahl der nebeneinanderliegenden Farb-Bits.
bits	Die geräteabhängigen Bits des Bitmap.

StretchBlt

Definition
```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Beschreibung Dieser für *StretchBlt* gespeicherte Record enthält ein geräteunabhängiges Bitmap, das auf allen Geräten abgespielt werden kann.

Elemente *rdSize:* Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x0B41.

rdParm: Enthält folgende Elemente:

Element	Beschreibung
raster op	Niederwertiges *WORD* der Raster-Operation.
raster op	Höherwertiges *WORD* der Raster-Operation.

Element	Beschreibung
SYE	y-Abmessung der Quelle.
SXE	x-Abmessung der Quelle.
SY	y-Koordinate des Ursprungs der Quelle.
SX	x-Koordinate des Ursprungs der Quelle.
DYE	y-Abmessung des Ziels.
DXE	x-Abmessung des Ziels.
DY	y-Koordinate des Ursprungs des Ziels.
DX	x-Koordinate des Ursprungs des Ziels.
BitmapInfo	*BITMAPINFO*-Struktur.
bits	Die geräteunabhängigen Bits des Bitmap.

StretchDIBits

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x0F43.

rdParm: Enthält folgende Elemente:

Element	Beschreibung
dwRop	Raster-Operation, die ausgeführt werden soll.
Usag	Flag, das anzeigt, ob die Bitmap-Farbtabelle RGB-Werte oder Indizes in die aktuell realisierte virtuelle Palette enthält.
srcYExt	Höhe der Quelle des Bitmap.
srcXExt	Breite der Quelle des Bitmap.
srcY	y-Koordinate des Ursprungs der Quelle des Bitmap.
srcX	x-Koordinate des Ursprungs der Quelle des Bitmap.
dstYExt	Höhe des Zielrechtecks.
dstXExt	Breite des Zielrechtecks.
dstY	y-Koordinate des Ursprungs des Zielrechtecks.
dstX	x-Koordinate des Ursprungs des Zielrechtecks.
BitmapInfo	*BITMAPINFO*-Struktur.
bits	Die geräteunabhängigen Bits des Bitmap.

TextOut

Definition

```
struct {
   DWORD rdSize;
   WORD  rdFunction;
   WORD  rdParm[];
}
```

Elemente

rdSize: Bestimmt die Record-Größe in *WORD*.

rdFunction: Bestimmt die GDI-Funktionsnummer 0x0521

rdParm: Enthält folgende Elemente:

Element	Beschreibung
count	Länge des String.
string	Der eigentliche String.
y_value	Virtuelle y-Koordinate des Startpunkts für den String.
x_value	Virtuelle x-Koordinate des Startpunkts für den String.

Index

A
ABC-Struktur 529
Advise-Transaktion, DDEML 52
Aktionsarten, OLE 66
Aktualisieren-Befehl, OLE-Server 103
Anwendungsname, DDE-Server 35
asynchrone Transaktion 53
Auschneiden-Befehl
  OLE-Server-Anwendungen 102f
ausführbare Strings
  internationale Execute-Befehle 116
  Syntax für Standardbefehle 116
Ausschneiden-Befehl
  OLE-Client-Anwendungen 91
  OLE-Server-Anwendungen 81

B
BN_HILITE-Benachrichtigungsbotschaft 525
BN_PAINT-Benachrichtigungsbotschaft 525
BN_UNHILITE-Benachrichtigungsbotschaft 525

C
CallBack-Funktion, OLECLIENTVTBL-Struktur 595
CB_ADDSTRING-Botschaft 493
CB_DELETESTRING-Botschaft 494
CB_FINDSTRINGEXACT-Botschaft 494
CB_GETDROPPEDCONTROLRECT-Botschaft 495
CB_GETDROPPEDSTATE-Botschaft 496
CB_GETEXTENDEDUI-Botschaft 496
CB_GETITEMHEIGHT-Botschaft 497
CB_SETEXTENDEDUI-Botschaft 498
CB_SETITEMHEIGHT-Botschaft 498
CBN_CLOSEUP-Benachrichtigungsbotschaft 525
CBN_SELENDCANCEL-Benachrichtigungsbotschaft 526
CBN_SELENDOK-Benachrichtigungsbotschaft 526
CBT_CREATEWND-Struktur 529
CBTACTIVATESTRUCT-Struktur 530
ChooseColor-Funktion 7f
CHOOSECOLOR-Struktur 6f, 530f
ChooseFont-Funktion 11
CHOOSEFONT-Struktur 11, 533, 535, 537, 539
CLASSENTRY-Struktur 540
Client-Anwendungen
  asynchrone Operationen 88
  Benutzerschnittstelle 78
  DDE in OLE-Anwendungen 110f, 113ff
  DDE-Transaktionen 34
  Dokumente schließen 87
  Dokumente speichern 87
  Einfügen-Befehl 96
  Klassenname-Objekt-Befehl 96
  Objekte anzeigen 90
  Objekte drucken 90
  Objekte erzeugen 93
  Objekte kopieren 91
  Objekte löschen 91
  Objekte öffnen und schließen 90
  OLE-Client-Anwendungen 70
  Rückgängig-Befehl 95
  schließen 97
  starten 85
  Verbunddokumente öffnen 86
  Verknüpfung-einfügen-Befehl 94
Close-Funktion, OLESERVERDOCVTBL-Struktur 606

COLOROKSTRING-Botschaft 703
CommDlgExtendedError-Funktion 31
COMSTAT-Struktur 540
CONVCONTEXT-Struktur 541
CONVINFO-Struktur 46, 542f
CPLINFO-Struktur 545
Create-Funktion,
  OLESERVERVTBL-Struktur 612
CreateFromTemplate-Funktion,
  OLESERVERVTBL-Struktur 613
CTLINFO-Struktur 546
CTLSTYLE-Struktur 547
CTLTYPE-Struktur 549

D
Datei-öffnen-Dialogfenster
  anzeigen 12
  Dateinamen überwachen 17
  Listen überwachen 16
Datei-speichern-unter-Dialogfenster
  anzeigen 15
  Dateinamen überwachen 17
  Listen überwachen 16
Daten-Handle
  dynamischer Datenaustausch 48
Datentypen 485
DDE
  Beschreibung 33
DdeAbandonTransaction-Funktion 54
DdeAccessData-Funktion
  Befehls-String 53
  globale Speicherobjekte 49
DDEACK-Struktur 550
DdeAddData-Funktion 50
DDEADVISE-Struktur 550
DdeCallback-Funktion 37
DdeClientTransaction-Funktion
  Advise-Transaktion 52
  asynchrone Transaktionen 54
  Execute-Transaktion 53
  Poke-Transaktion 51
  Request-Transaktion 51
  synchrone Transaktionen 54
DdeConnect-Funktion 43
DdeConnectList-Funktion 46
DdeCreateDataHandle-Funktion 48
DdeCreateStringHandle-Funktion 39
DDEDATA-Struktur 551
DdeDisconnect-Funktion 45, 47
DdeDisconnectList-Funktion 47
DdeEnableCallback-Funktion 55
DdeFreeDataHandle-Funktion 50
DdeFreeStringHandle-Funktion 41
DdeGetData-Funktion 50
DdeInitialize-Funktion
  DDE-Anwendungen überwachen 58
  DDEML initialisieren 37
DdeKeepStringHandle-Funktion 41
DDEML
  Advise-Transaktion 52
  asynchrone Transaktionen 53
  Beschreibung 33
  Callback-Funktion 38
  Client- und Server-Interaktion 34
  Datenverwaltung 48
  Einzelkonversationen 43
  Elementname 35
  Execute-Transaktion 53
  Fehlererkennung 58
  initialisieren 37
  Konversationen beenden 38
  Mehrfachkonversationen 46
  OLE mit DDEML benutzen 70
  Poke-Transaktion 51
  Request-Transaktion 51
  Servicenamen 35
  Servicenamen registrieren 41
  Servicenamenfilter 42
  String-Verwaltung 39
  synchrone Transaktionen 53
  Thema System 35
  Themenname 35
  Transaktionen 35
  Transaktionen sperren 55
  Transaktionsklassen 55
  Transaktionssteuerung 55
  Transaktionstypen 56
  Transaktionsverwaltung 50
  überwachende Anwendungen 58
  und OLE 67
DdeNameService-Funktion 41f
DDEPOKE-Struktur 552
DdePostAdvise-Funktion 52
DdeQueryConvInfo-Funktion 46, 54
DdeQueryNextServer-Funktion 46
DdeQueryString-Funktion 40
DdeReconnect-Funktion 46
DdeSetUserHandle-Funktion 54
DdeUnaccessData-Funktion 49
DdeUninitialize-Funktion 38
DEBUGHOOKINFO-Struktur 553
DECLARE_HANDLE-Makro 665

DECLARE_HANDLE32-Makro 665
DefLoadFromStream-Funktion 109
DEVNAMES-Struktur 554
DllCreateFromFlip-Funktion 108
DllLoadFromStream-Funktion 109
DOCINFO-Struktur 555
DoVerb-Funktion,
  OLEOBJECTVTBL-Struktur 601
DRIVERINFOSTRUCT-Struktur 555
Drucken-Dialogfenster 18f
Drucker, Standarddrucker 19
Druckereinrichtung-Dialogfenster 18
DRVCONFIGINFO-Struktur 556
dynamischer Datenaustausch (DDE)
  ausführbare Strings 116
  Elemente des OLE-Themas System 113
  OLE-Bibliotheken und Standard-DDE-Operationen 68
  OLE-Client-Anwendungen 110
  OLE-Konversationen 113
  OLE-Server-Anwendungen 112
  OLE-Standardelementnamen 114

E
Edit-Funktion,
  OLESERVERVTBL-Struktur 614
Einfügen-Befehl, OLE-Anwendungen 94
Einfügen-mit-Formatauswahl-Befehl, OLE 95
eingebettetes Objekt 64
Elementname, DDE-Server 35
EM_GETFIRSTVISIBLELINE-Botschaft 499
EM_GETPASSWORDCHAR-Botschaft 500
EM_GETWORDBREAKPROC-Botschaft 500
EM_SETREADONLY-Botschaft 501
EM_SETWORDBREAKPROC-Botschaft 501
EnumClipboardFormats-Funktion 95
Ersetzen-Dialogfenster
  anzeigen 22
  Botschaften bearbeiten 23
EVENTMSG-Struktur 556
Execute-Funktion
  OLESERVERDOCVTBL-Struktur 610
  OLESERVERVTBL-Struktur 616
Execute-Transaktion, DDEML 53
Exit-Funktion,
  OLESERVERVTBL-Struktur 615

F
Farben-Dialogfenster
  Beschreibung 3
  Grundfarben anzeigen 6
  HSL-Farbmodell 5
  RGB-Farbmodell 4
  selbstdefinierte Farben 7
Fehlererkennung
  DDEML 58
  Standarddialogfenster 31
FIELDOFFSET-Makro 665
FILEOKSTRING-Botschaft 704
Filterfunktion
  Standarddialogfenster 25
FINDMSGSTRING-Botschaft 23, 704
FINDREPLACE-Struktur 21f, 557, 559, 561
FindText-Funktion 21
FIXED-Struktur 562
FMS_GETDRIVEINFO-Struktur 562
FMS_GETFILESEL-Struktur 563
FMS_LOAD-Struktur 563
Funktionen
  asynchrone OLE-Operationen 89
  DdeCallback-Funktion 37
  OLE-Dokumente verwalten 86
  OLE-Objekt-Handler 107
  OLE-Objekte erzeugen 93
  OLE-Server-Anwendungen 98

G
Get-Funktion,
  OLESTREAMVTBL-Struktur 618
GetBValue-Makro 666
GetData-Funktion 69
  OLEOBJECTVTBL-Struktur 601
GetGValue-Makro 666
GetObject-Funktion 69
  OLESERVERDOCVTBL-Struktur 608
GetOpenFileName-Funktion 12
GetRValue-Makro 667
GetSaveFileName-Funktion 15
GetWinFlags-Funktion
  DDEML initialisieren 37
GLOBALENTRY-Struktur 564f
GLOBALINFO-Struktur 567
GLYPHMETRICS-Struktur 568

H
HARDWAREHOOKSTRUCT-Struktur 568
HELPMSGSTRING-Botschaft 705
HELPWININFO-Struktur 569

Hilfe-Schalter
  Standarddialogfenster 30
HSL-Farbmodell 5
HSZPAIR-Struktur 44, 570

K
KERNINGPAIR-Struktur 570
Klassenname-Objekt-Befehl, OLE-Anwendungen 96
Kopieren-Befehl
  OLE-Client-Anwendungen 91
  OLE-Server-Anwendungen 81, 102f

L
LB_FINDSTRINGEXACT-Botschaft 502
LB_GETCARETINDEX-Botschaft 503
LB_SETCARETINDEX-Botschaft 503
LBN_SELCANCEL-Benachrichtigungsbotschaft 527
LBSELCHSTRING-Botschaft 705
LoadString-Funktion 14
LOCALENTRY-Struktur 570f, 573
LOCALINFO-Struktur 574
LOGFONT-Struktur
  Schriftart-Dialogfenster 11
  TrueType-Schriften 103

M
MAKELP-Makro 667
MAKELPARAM-Makro 667
MAKELRESULT-Makro 668
MAT2-Struktur 574
MEMMANINFO-Struktur 575
Metafile, OLE-Server-Anwendungen 103
METAHEADER-Struktur 576
METARECORD-Struktur 577
MINMAXINFO-Struktur 577
MODULEENTRY-Struktur 578
MONCBSTRUCT-Struktur 578
MONCONVSTRUCT-Struktur 579
MONERRSTRUCT-Struktur 580
MONHSZSTRUCT-Struktur 581
MONLINKSTRUCT-Struktur 582
MONMSGSTRUCT-Struktur 583
MOUSEHOOKSTRUCT-Struktur 584
MOUSETRAILS-Escape-Code 671

N
Native-Format, Zwischenablage 72
NCCALCSIZE_PARAMS-Struktur 584
NEWCPLINFO-Struktur 585
NEWTEXTMETRIC-Struktur 586f, 589
NFYLOADSEG-Struktur 591
NFYLOGERROR-Struktur 592
NFYLOGPARAMERROR-Struktur 592
NFYRIP-Struktur 593
NFYSTARTDLL-Struktur 594

O
ObjectLink-Format, Zwischenablage 72
ObjectLong-Funktion, OLEOBJECTVTBL-Struktur 603
Objekt-einfügen-Befehl 94
Objekt-Handler
  Beschreibung 71
  implementieren 106
  Objekte erzeugen 108
OFFSETOF-Makro 668
OLE
  Aktionsarten 66
  asynchrone Operationen 88
  ausführbare DDE-Strings 116
  DDE-Client-Anwendungen 110f, 113, 115, 117, 119
  DDE-Konversationen 113
  DDE-Operationen mit OLE 68
  DDE-Server-Anwendungen 112
  DDE-Standardelementnamen 114
  DDE-Thema System 113
  DDEML mit OLE benutzen 70
  DDEML und OLE vergleichen 67
  eingebettete Objekte 64
  Klassenname-Objekt-Befehl 96
  Objekt-Handler implementieren 106
  Objekte mit Objekt-Handler erzeugen 108
  OLECLI.DLL 71
  OLESVR.DLL 71
  Pakete 65
  Speicherformate für Objekte 82
  verknüpftes Objekte 64
  Vorteile 66
OLE-Client-Anwendungen
  Beschreibung 84f, 87, 89, 91
  Dokumente schließen 87
  Dokumente speichern 87
  Dokumente verwalten 86
  Einfügen- und Verknüpfung-einfügen-Befehl 94
  Objekte anzeigen 90
  Objekte drucken 90

Objekte erzeugen 93
Objekte kopieren 91
Objekte löschen 91
Objekte öffnen und schließen 90
Rückgängig-Befehl 95
schließen 97
starten 85
OLE-Datentransfer
Benutzerschnittstelle der Client-Anwendung 78
Benutzerschnittstelle der Server-Anwendung 81
Client-Anwendungen 70
Kommunikation zwischen Bibliotheken 71
neue und geänderte Befehle 78
Objekt-Handler 71
Pakete 80
Registrierungsdatenbank 75
Server-Anwendungen 70
Versionskontrolle für Server 77
Zwischenablagekonventionen 72
OLE-Server-Anwendungen
Aktualisieren-Befehl 103
Ausschneiden- und Kopieren-Befehl 102f
Dokumente öffnen und schließen 101
Funktionen 98
schließen 104
Speichern-Befehl 103
starten 99
OLE-Verbunddokumente
Beschreibung 63
Illustration 64
öffnen 86
OleActivate-Funktion
Klassenname-Objekt-Befehl - implementieren 96
Objekte öffnen 91
OleBlockServer-Funktion
asynchrone Operationen 88
Warteschlange für Botschaften 101
OLECLIENT-Struktur 594
Client-Anwendungen starten 85
Objekt-Handler 107
Verbunddokumente öffnen 86
OLECLIENTVTBL-Struktur 85f, 595
OleClone-Funktion
Objekte in die Zwischenablage kopieren 92
Rückgängig-Befehl 95
OleClose-Funktion 91
OleCopyFromLink-Funktion 94
OleCopyToClipboard-Funktion 79, 91
OleCreate-Funktion 94
OleCreateFromClip-Funktion
Client-Anwendungen 95
Objekt-Handler 108
OleCreateFromFile-Funktion 69
OleCreateFromTemplate-Funktionen 94
OleCreateLinkFromClip-Funktion 95
OleDelete-Funktionen 91
OleDraw-Funktion 90
OleEnumFormats-Funktion 92
OleGetData-Funktion 97
OleGetLinkUpdateOptions-Funktion 96
OleLoadFromStream-Funktion 86
OLEOBJECT-Struktur 597
Client-Anwendungen 93
Objekt-Handler 107
Objekte öffnen 102
Server-Anwendungen starten 100
OleObjectConvert-Funktion 97
OLEOBJECTVTBL-Struktur 100, 598f, 601, 603
OleQueryBounds-Funktion 90
OleQueryCreateFromClip-Funktion 94
OleQueryLinkFromClip-Funktion 95
OleQueryOpen-Funktion 91
OleQueryReleaseError-Funktion
Client-Anwendung schließen 98
Objekte erzeugen 94
OleQueryReleaseMethod-Funktion 98
OleQueryReleaseStatus-Funktion
asynchrone Operationen 89
Client-Anwendungen schließen 98
Objekte aktivieren 90
OleQuerySize-Funktion 87
OleQueryType-Funktion 95
OleReconnect-Funktion 91
OleRegisterClientDoc-Funktion 86, 92
OleRegisterServer-Funktion
DDE-Operationen 69
Server-Anwendungen starten 99
OleRegisterServerDoc-Funktion
DDE-Operationen 69
Dokumente oder Objekte öffnen 101
Server-Anwendungen starten 99
OleRelease-Funktion
Client-Anwendungen schließen 97
Dokumente schließen 87
Objekte schließen 91

OleRenameServerDoc-Funktion 103
OleRequestData-Funktion 76
OleRevertClientDoc-Funktion 88
OleRevokeClientDoc-Funktion 88, 92
OleRevokeObject-Funktion 105
OleRevokeServerDoc-Funktion 105
OleSavedClientDoc-Funktion 88, 92
OleSavedServerDoc-Funktion 103
OleSaveToStream-Funktion 87, 92
OLESERVER-Struktur 605
  Objekt-Handler 107
  Server-Anwendungen starten 100
OLESERVERDOC-Struktur 605
  Dokumente öffnen 101
  Objekt-Handler 107
OLESERVERDOCVTBL-Struktur 605, 607, 609
  DDE-Operationen 69
  Server-Anwendungen starten 100
OLESERVERVTBL-Struktur 611, 613, 615
  Dokumente aktualisieren 103
  Dokumente oder Objekte öffnen 101
  Server-Anwendungen schließen 104
  Server-Anwendungen starten 100
OleSetBounds-Funktion 90
OleSetData-Funktion
  Datenformate registrieren 76
  DDE-Operationen 69
  Verknüpfungen bearbeiten 97
OleSetHostNames-Funktion 91
OleSetLinkUpdateOptions-Funktion 96
OleSetTargetDevice-Funktion 90
OLESTREAM-Struktur 617
  Client-Anwendungen starten 85
  Verbunddokumente öffnen 86
OLESTREAMVTBL-Struktur 85f, 617
OLETARGETDEVICE-Struktur 619
OleUnblockServer-Funktion 101
OleUpdate-Funktion
  Objekte anzeigen 90
  Verknüpfung aktualisieren 97
Open-Funktion,
  OLESERVERVTBL-Struktur 611
OPENFILENAME-Struktur 620f, 623, 625, 627
  Datei-öffnen-Dialogfenster 12
  Datei-speichern-unter-Dialogfenster 15
OUTLINETEXTMETRIC-Struktur 628f
OwnerLink-Zwischenablageformat 72

P

Pakete
  OLE-Anwendungen 65, 80
PANOSE-Struktur 632f, 635
POINTFX-Struktur 636
Poke-Transaktion, DDEML 51
POSTSCRIPT_DATA-Escape-Code 672
POSTSCRIPT_IGNORE-Escape-Code 672
PrintDlg-Funktion 19
PRINTDLG-Struktur 19, 637, 639, 641, 643
Put-Funktion,
  OLESTREAMVTBL-Struktur 618

R

RASTERIZER_STATUS-Struktur 645
RegisterClipboardFormat-Funktion 77, 100
RegisterWindowMessage-Funktion
  Datei-öffnen-Dialogfenster 14
  Dateinamen überwachen 17
  Farben-Dialogfenster 9
  Hilfe-Schalter in Standarddialogfenstern 30
  Listen überwachen 17
  Suchen- und Ersetzen-Dialogfenster 23
Registrierungsdatenbank
  OLE-Anwendungen 75
Release-Funktion
  OLEOBJECTVTBL-Struktur 600
  OLESERVERDOCVTBL-Struktur 609
  OLESERVERVTBL-Struktur 615
ReplaceText-Funktion 22
Request-Transaktion, DDEML 51
RGB-Farbmodell 4
Rückgängig-Befehl,
  OLE-Anwendungen 95

S

Save-Funktion,
  OLESERVERDOCVTBL-Struktur 606
Schablone
  Standarddialogfenster 28
SearchFile-Funktion 23
SEGINFO-Struktur 645
SELECTOROF-Makro 669
Server-Anwendungen
  Aktualisieren-Befehl 103
  Ausschneiden- und Kopieren-Befehl 102f
  Benutzerschnittstelle 81
  DDE-Transaktionen 35
  Dokumente oder Objekte öffnen 101

notwendige DDE-Befehle 116
OLE-Funktionen 98
OLE-Server 70
OLE-Server und DDE 112
schließen 104
Speichern-Befehl 103
starten 99
Versionskontrolle 77
Servicename
DDE-Server 35
SETALLJUSTVALUES-Escape-Code 673
SetClipboardData-Funktion
Client-Anwendungen 92
Server-Anwendungen 102
SetColorScheme-Funktion
OLEOBJECTVTBL-Struktur 604
OLESERVERDOCVTBL-Struktur 609
SetData-Funktion,
OLEOBJECTVTBL-Struktur 602
SetDocDimensions-Funktion,
OLESERVERDOCVTBL-Struktur 607
SetHostNames-Funktion,
OLESERVERDOCVTBL-Struktur 607
SETRGBSTRING-Botschaft 706
SetTargetDevice-Funktion,
OLEOBJECTVTBL-Struktur 602
SHAREVISTRING-Botschaft 707
SHELL.DLL
OLE-Anwendungen 75
Show-Funktion,
OLEOBJECTVTBL-Struktur 600
SIZE-Struktur 647
Speichern-unter-Befehl, OLE-Server 103
STACKTRACEENTRY-Struktur 647
Standarddialogfensterbibliothek
Beschreibung 1
COMMDLG.DLL 1
Datei-öffnen-Dialogfenster
anzeigen 12
Datei-speichern-unter-Dialogfenster
anzeigen 15
Dateinamen überwachen 17
Dialogfenster anpassen 28
Dialogfensterschablone 28
Drucken-Dialogfenster 18
Druckereinrichtung-Dialogfenster 18
Ersetzen-Dialogfenster 20
Farben-Dialogfenster 3
Fehlerüberwachung 31
Filterfunktion 25
Grundfarben anzeigen 6
Hilfe-Schalter 30
HSL-Farbmodell 5
Listen überwachen 16
RGB-Farbmodell 4
Schriftart-Dialogfenster 9
selbstdefinierte Farben 7
Standarddialogfenster anpassen 24
Suchen-Dialogfenster 20
STM_GETICON-Botschaft 504
STM_SETICON-Botschaft 504
String-Handle, DDE 39
Suchen-Dialogfenster
anzeigen 21
Botschaften bearbeiten 23
synchrone Transaktionen, DDEML 53
SYSHEAPINFO-Struktur 648
System
Thema 35

T

TASKENTRY-Struktur 649
Thema
Name 35
System, DDE und OLE 113
System, DDEML 35
TIMERINFO-Struktur 650
Transaktionen, DDE 34
TrueType-Schriften,
Server-Anwendungen 103
TTPOLYCURVE-Struktur 650
TTPOLYGONHEADER-Struktur 651

U

überwachende Anwendungen 58

V

Verbunddokumente
Beschreibung 63
öffnen 86
verknüpftes Objekt 64
Verknüpfung-einfügen-Befehl,
OLE-Anwendungen 94
Versionskontrolle für OLE-Server 77
VS_FIXEDFILEINFO-Struktur 652f, 655

W

WINDEBUGINFO-Struktur 656f, 659
WINDOWPLACEMENT-Struktur 660f
WINDOWPOS-Struktur 662f
Windows-Datentypen 485
WinHelp-Funktion 31

WM_CHOOSEFONT_GETLOGFONT-Botschaft 12, 505
WM_COMMNOTIFY-Botschaft 506
WM_DDE_ACK-Botschaft 507
WM_DDE_ADVISE-Botschaft 69, 509
WM_DDE_DATA-Botschaft 510f
WM_DDE_EXECUTE-Botschaft 512
WM_DDE_INITIATE-Botschaft 513
WM_DDE_POKE-Botschaft 69, 515
WM_DDE_REQUEST-Botschaft 516
WM_DDE_TERMINATE-Botschaft 517
WM_DDE_UNADVISE-Botschaft 518
WM_DROPFILES-Botschaft 94, 519
WM_INITDIALOG-Botschaft 24
WM_PALETTEISCHANGING-Botschaft 519
WM_POWER-Botschaft 520
WM_QUEUESYNC-Botschaft 521
WM_SYSTEMERROR-Botschaft 521
WM_USER-Botschaft 522
WM_WINDOWPOSCHANGED-Botschaft 523
WM_WINDOWPOSCHANGING-Botschaft 524

X

XTYP_ADVDATA-Transaktion 675
XTYP_ADVREQ-Transaktion 676
XTYP_ADVSTART-Transaktion 52, 677
XTYP_ADVSTOP-Transaktion 53, 677
XTYP_CONNECT-Transaktion 43, 678
XTYP_CONNECT_CONFIRM-Transaktion 43f, 679
XTYP_DISCONNECT-Transaktion 46f, 680
XTYP_ERROR-Transaktion 681
XTYP_EXECUTE-Transaktion 53, 681
XTYP_MONITOR-Transaktion 59, 682
XTYP_POKE-Transaktion 51, 683
XTYP_REGISTER-Transaktion 41, 684
XTYP_REQUEST-Transaktion 40, 51, 685
XTYP_UNREGISTER-Transaktion 686
XTYP_WILDCONNECT-Transaktion 46, 686f
XTYP_XACT_COMPLETE-Transaktion 54, 688f

Z

Zwischenablage
  Formate 72
  OLE-Konventionen 72

ADDENDUM

BORLAND® C++

BORLAND

BORLAND GMBH, GAUTINGER STR. 10,POSTFACH 1209, D-8130 STARNBERG, TELEFON 0 81 51 / 263-0, TELEFAX 0 81 51 / 142 09
CORPORATE HEADQUARTERS: 1800 GREEN HILLS ROAD, P.O. BOX 660001, SCOTTS VALLEY, CA 95067-0001. (408) 438-530
OFFICES IN: AUSTRALIA, DENMARK, FRANCE, GERMANY, ITALY, JAPAN, NEW ZEALAND, SINGAPORE, SWEDEN AND THE UNITED KINDOM * ARTIKEL-NR. 147313